UNIVERSIDAD AUTONOMA DE MADRID

ESCUELA POLITECNICA SUPERIOR

Doble Grado en Ingeniería Informática y Matemáticas

TRABAJO FIN DE GRADO

Sistema adaptativo de prevención de intrusos mediante Honeypots

Autor: Daniel Mayordomo Trujillano
Tutor: Francisco de Borja Rodríguez Ortiz

JULIO 2016

Sistema adaptativo de prevención de intrusos mediante Honeypots

AUTOR: Daniel Mayordomo Trujillano
TUTOR: Francisco de Borja Rodríguez Ortiz

Dpto. Ingeniería Informática

Escuela Politécnica Superior
Universidad Autónoma de Madrid

Julio de 2016

Resumen

No cabe duda que, en la actualidad, la seguridad informática es un aspecto que preocupa a millones de usuarios de la red, bien sea para mantener la confidencialidad de su información, su integridad o su disponibilidad.

Los Honeypots son herramientas que permiten recopilar y analizar información sobre las diferentes amenazas que sufren los sistemas durante todo su ciclo de vida. Para conseguir esta información simulan pequeños sistemas informáticos vulnerables que serán un blanco fácil para los atacantes.

Si bien estas herramientas llevan utilizándose durante años, el tratamiento de los datos que pueden recabar se encuentra en una fase inicial ya que no se realiza un análisis profundo de éstos y no se emplean para impedir ataques futuros.

Esta investigación se centra en analizar y estudiar los ataques de los Honeypots instalados para la creación automática y dinámica de reglas que permitan mejorar la seguridad en nuestro sistema informático. Previamente, se realiza un estudio de los Honeypots del que se extraen tres objetivos bien diferenciados. El primero es el estudio de las herramientas de Honeypot utilizadas en la actualidad en una distribución de máquina virtual. En segundo lugar, se pretende recolectar información de una serie de Honeypots instalados en la UAM para este propósito. En tercer lugar, se realizará una transformación de la información recogida por los Honeypots ubicados en la red anteriormente mencionada, en las reglas de bloqueo que se añadirán al sistema con el fin de mejorar su seguridad.

Para llevar a cabo este proyecto y poder alcanzar los objetivos marcados se ha creado una infraestructura que recoge información de los ataques sufridos por los Honeypots para su posterior procesamiento. Este programa, 'HoneyPRules.py', utiliza los Honeypots para la creación de reglas de bloqueo de una forma automática y dinámica utilizando Iptables. Este programa permite la interactuación entre la infraestructura y un usuario final de una forma sencilla e intuitiva, posibilitando que el usuario pueda crear una configuración personalizada de las reglas de su sistema.

De esta investigación se desprende que las reglas creadas e incorporadas al sistema por la infraestructura son efectivas. Por ello, hacen que la seguridad de éste aumente debido a que los Honeypots de la infraestructura, al estar continuamente expuestos a ataques, detecten direcciones altamente peligrosas de una forma más rápida, pudiendo así proteger al sistema antes de ser atacado.

Palabras clave

Honeypot, Honeynet, Kippo, Glastopf, Seguridad Informática, Informática Forense, Taxonomía Honeypot, Taxonomía Ataques Informáticos, Ataques Informáticos, IDS, Firewall, Iptables, Virtualización.

Abstract

There is no doubt that computer security, nowadays, is an issue of concern to millions of network users, either to maintain the confidentiality of their information, integrity or availability.

Honeypots are tools to collect and analyze information about the various threats to systems throughout their life cycle. To get this information Honeypots simulate small vulnerable computer systems that will be an easy target for attackers.

Although these tools have been used for many years, the treatment of the data collected by them is at an early stage since no in-depth-analysis has been made by in order to prevent future attacks.

This research focuses on analyzing and studying the attacks on the installed Honeypots for automatic and dynamic creation of rules to improve safety in our computer system. We can extract three distinct objectives from a previous research of the Honeypots. The first of them is the study of honeypot tools currently used in the virtual machines. Secondly, it aims at the collection information from different installed Honeypots at UAM for this purpose. Thirdly, a transformation of the information collected, located in the aforementioned network, to create blocking rules to will be added to the system in order to improve their security.

To carry out this project and to achieve the objectives, an infrastructure has been created in order to collect information from the attacks on the Honeypots for further processing. This program, 'HoneyPRules.py', uses the Honeypots to create automatic and dynamic locking rules using Iptables. This program allows the interaction between the infrastructure and an end user in a simple and intuitive way, enabling the user to create a custom configuration of his/her system rules.

This investigation shows that the created rules which were incorporated into the system by the infrastructure are effective. Therefore, they increase the security because the Honeypots installed in the infrastructure are constantly exposed to attacks, detecting highly dangerous addresses more quickly, which can protect the system before being attacked.

Keywords

Honeypot, Honeynet Kippo, Glastopf, Computer Segurity, Forensic Computer Science, Taxonomy Honeypot, Taxonomy Computer Science Attacks, Computer Science Attacks, IDS, Firewall, Iptables, Virtualization.

Agradecimientos

Esta parte de la memoria parece la menos importante pero en realidad es una de las más importantes, ya que al finalizar tu carrera y al hacer tu TFG puedes mostrar a aquellas personas que han estado contigo a lo largo de todos los años tu gratitud por ello.

Agradezco a todas aquellas personas que han estado conmigo en la carrera y han contribuido a que yo pueda estar aquí ahora mismo presentando el TFG. En primer lugar a mi tutor Francisco de Borja Rodríguez Ortiz por ayudarme durante la realización de este trabajo y respondiendo todas mis dudas relacionadas con éste, y a Diego Jurado por ayudarme a que este trabajo fuese por buen camino.

Agradecer a todos aquellos compañeros de clase con los que he estado durante estos 'cortos' años de carrera y que hemos pasado tiempo juntos. No mencionaré nombres concretos en este apartado porque aquellos compañeros que he tenido y que luego les tengo como amigos de verdad saben perfectamente quiénes son y saben que les aprecio en gran medida y les agradezco de corazón todo lo que han hecho y siguen haciendo por mí.

Agradecer a todos mis amigos de mi pueblo, Santa Lucía de la Sierra, porque en los periodos vacacionales han hecho que me pudiese relajar y pasármelo bien.

Por último, mi más profundo agradecimiento a mi familia por haber estado conmigo en todo momento y aguantarme durante estos años, que sé en algunos momentos les transmitía mis nervios y ansiedad.

A todos, buenos días y muchas gracias.

INDICE DE CONTENIDOS

INDICE DE FIGURAS

INDICE DE TABLAS

1. Introducción

El presente proyecto se basa en el estudio de la mejora de la seguridad en las redes y sistemas informáticos presentes en nuestro día a día, el cual forma parte del área de la Seguridad Informática e Informática Forense. Existen diferentes proyectos cuyas líneas de investigación son similares, con el fin de ayudar a la innovación y mejora de los sistemas de protección de nuestros equipos informáticos. Una de estas herramientas, y en la que vamos a centrar nuestro trabajo, es la denominada Honeypot [1][2][5], que ayuda a recopilar y analizar información de las diferentes amenazas que sufren los sistemas durante su ciclo de vida, tales como ataques por ssh, virus, inyección de SQL y muchos más, gracias a que simulan pequeños sistemas informáticos vulnerables y por tanto, más promiscuos a ser atacados por agentes externos de la red. Por este motivo se han utilizado diferentes tipos de estas herramientas para conseguir información de los ataques que reciben para posteriormente crear reglas que se incorporan a nuestro sistema para mejorar su seguridad, todo ello con el fin de estudiar su impacto sobre los sistemas.

En este proyecto el objetivo principal es utilizar la información que recopilan los Honeypots para mejorar la seguridad de un sistema informático mediante la creación automática de reglas, además de un método de investigación y estudio de ella para ver su efectividad.

Debido a la incorporación de las tecnologías de la información en diferentes ámbitos de la vida cotidiana, así como en todo aquello que se está creando para el avance en diferentes disciplinas científicas, hacen que uno de los principales objetivos que se marca la Informática y aquellas ramas de la ciencias relacionadas con ella, es la mejora e investigación en la seguridad informática. Actualmente la información que manejan grandes empresas, como pequeños comercios o trabajadores son de tipo informático, es decir, son almacenados en diferentes dispositivos informáticos para su posterior uso, como por ejemplo una memoria USB, un disco duro externo, un teléfono móvil, etc. Esto obliga a la informática forense a estar en continuo avance e investigación de los sistemas atacantes que almacenan datos de terceros para su uso en beneficio propio. Por este motivo la herramienta Honeypot es de gran ayuda ya que simula un sistema muy vulnerable para ver las estrategias creadas por los sistemas atacantes y así poder detectarlas en el futuro.

Como resultado de todo lo mencionado anteriormente, la informática forense tiene el reto de mantenerse actualizada en los nuevos artificios que crean los atacantes para así poder recopilar la información necesaria de ellos y utilizarla de forma adecuada para mejorar la seguridad de los sistemas informáticos.

Tras barajar distintas formas de instalación de la herramienta Honeypot en sus diversos tipos [6] se decidió diseñar, crear y configurar una red propia de éstos, Honeynet [3][4][5], colocando en diferentes direcciones IP varios tipos de Honeypot con diferentes singularidades para recopilar una mayor y variada cantidad de datos. Se descartó la instalación de sistemas ya montados con Honeypots, como lo es Honeydrive [7], debido a que no contemplaban algunas funcionalidades necesarias en nuestro proyecto, y para así poder llevar a cabo el proceso completo de estudio de Honeypots. Todo esto se realizó en colaboración con otro compañero, Diego Jurado Pallares – quien en su TFG ha realizado una infraestructura para integrar todos los datos que analizará en profundidad con posterioridad.

Tras haber decidido el modelo de organización de los Honeypots, se realizó un análisis de los ataques recibidos en éstos y se instalaron aquellos tipos de Honeypots que se enfocaban a los ataques valorados como más relevantes. Finalmente, se decidió utilizar 2 parejas de los Honeypots Cowrie [8] y Glastopf [9] en una misma dirección IP, ya que no generaban

interferencias en sus ejecuciones, ambas instaladas en un sistema Debian, debido a las facilidades que ofrecía para la configuración de ambos Honeypots.

Seguidamente, tras la puesta en marcha de los Honeypots, se procedió a detectar los diferentes tipos de ataques que recogían. Para comprobar que la instalación se había realizado correctamente se llevaron a cabo ataques controlados a las herramientas y se almacenaron en las diferentes bases de datos creadas con anterioridad. Tras la verificación del despliegue adecuado de los Honeypots, se tomó la decisión de almacenar la información de todas las herramientas de forma centralizada en una base de datos, situada en un servidor fuera de la red donde se preparó todo, teniendo como resultado un mejor acceso y manejo de esta base de datos.

Posteriormente, se realizó un estudio de la información almacenada en la base de datos centralizada para poder crear las diferentes reglas automáticamente que se podrían incorporar en el Iptables de nuestro sistema para que la seguridad de éste mejore. Concretamente, se han creado tres tipos de reglas como una forma de afrontar este problema. La primera, TMP, ha sido asignada a aquellos atacantes que no superan el umbral de ataques máximos permitidos por la infraestructura creada por SSH. La segunda, M1, se ha utilizado para caracterizar a aquellos atacantes que han superado el umbral máximo de ataques permitidos por la infraestructura creada por SSH. Por último, la tercera, M2, es la encargada de designar a aquellos atacantes que han superado el umbral de ataques permitidos por la infraestructura creada por SSH y han realizado inyección SQL. Con el fin de poder mantener un backup de la información generada así como tener la posibilidad de debuguear nuestro programa 'HoneyPRules.py' se ha creado y mantenido un log que incluye, entre otra, la siguiente información: dirección IP a bloquear y tipo de regla.

Por último, se ha realizado un análisis de los resultados tras la integración de las reglas anteriormente creadas para comprobar si la seguridad de nuestro sistema ha aumentado. Se creó una infraestructura para la muestra de información integrada sobre los ataques y posterior creación e incorporación de reglas.

1.1 Motivación

El objetivo principal del proyecto es llevar a cabo un trabajo relacionado con el área de la herramienta Honeypot y la Informática Forense. Tras comprobar que los sistemas informáticos cada vez están más presentes en nuestra vida, que la seguridad de éstos es primordial para que la información, tanto personal como profesional no pueda ser accesible ni distribuida por terceros para su propio beneficio, y, debido a que el uso de Internet para cualquier tipo de transacción entre dos ordenadores no se hace físicamente, nos lleva a que los ataques informáticos sean un gran punto de interés para ayudar a que éstos disminuyan.

Aunque las herramientas Honeypot han ido actualizándose durante los años, siguen en una posición bastante alejada en cuanto al análisis y empleo de los datos que éstas recogen puesto que podrían ser utilizados para mejorar la seguridad de los equipos en los que se encuentren instaladas. Por todo ello surge la motivación de crear una infraestructura que permita recopilar información de los ataques que se reciban en los Honeypots para posteriormente procesarlos y emplear esta información para crear reglas que serán de utilidad en las denominadas Iptables. Aunque se dará una explicación más extensa sobre estas tablas más adelante, por el momento, diremos que contienen una serie de reglas a aplicar cuando se reciben nuevos paquetes a través de la red en un sistema informático con el fin de decidir sobre el tratamiento del paquete. Debido a las mejoras que implementan los atacantes de forma continua, es necesario mantener este sistema actualizado, tarea que se realiza

paralelamente debido al almacenamiento de la información de los ataques en los Honeypots para su posterior análisis.

1.2 Objetivos

El principal objetivo de este proyecto es, analizar y estudiar los ataques de los Honeypots instalados para la creación automática y dinámica de reglas en el Iptables que permitan mejorar la seguridad en nuestro sistema informático.

A continuación se exponen otros objetivos que se pretenden satisfacer a lo largo de este trabajo, así como las áreas a las cuales pertenecen cada uno de ellos:

- **Objetivo 1: Estudio de las herramientas Honeypot actuales en una distribución de máquina virtual (VM VirtualBox).**
  Este objetivo se basa en el estudio previo a realizar de la herramienta Honeypot para clarificar qué tipos de Honeypots son los que mejor nos ayudarán llevar a cabo la realización de nuestro trabajo.

  Para la realización de este objetivo se han seguido los siguientes pasos:
  - Se procederá a un estudio de las diferentes herramientas de Honeypots que ya existen y se comprobará cuáles de ellos se ajustan mejor al problema que se pretende resolver.
  - Se estudiarán y analizarán los ataques recibidos en las herramientas instaladas de forma local
  - Se configurarán e instalarán aquellos Honeypots que mejor información sobre los ataques recibidos por éstos nos ofrezcan en una red real.

- **Objetivo 2: Configuración e instalación definitivas de los diferentes Honeypots en una red de la UAM.**
  Se realizará el despliegue de aquellos Honeypots seleccionados en la fase de estudio en una red de la UAM.

- **Objetivo 3: Transformación de la información recogida por los Honeypots en reglas.**
  La información recopilada por los Honeypots se procesará y será utilizada para la creación de reglas al momento que serán implantarlas en Firewalls para aumentar la seguridad del sistema. Todo ello se llevarán a cabo por un programa creado tras el análisis de la información.

1.3 Metodología y planificación del trabajo

Este documento contiene las siguientes secciones o apartados:

- **Introducción.** Presentaremos el proyecto realizado por medio de una introducción, mostrando de dónde parte el trabajo, la motivación de su realización y los objetivos que se quieren conseguir.
- **Estado del arte.** Expondremos las bases teóricas en las cuales se apoya el trabajo para su realización y desarrollo.
- **Análisis.** Mostramos el estudio seguido para decidir la mejor opción que ayudará a desarrollar nuestro proyecto y el esquema general que se ha tomado como punto principal.

- **Diseño.** Llevaremos a cabo el esquema del proyecto así como la organización y división de las diferentes partes que contiene.
- **Desarrollo.** Explicaremos cómo se ha realizado el proyecto para poder llegar a los objetivos marcados al principio.
- **Integración, pruebas y resultados.** Plantearemos ejemplos para comprobar el correcto funcionamiento de nuestro trabajo.
- **Conclusiones, discusión y trabajo futuro.** Expondremos las conclusiones a las que se han llegado al finalizar el proyecto y plantearemos las diferentes formas de innovar en este proyecto en el futuro.

En cada una de ellas se expondrá con mayor detenimiento toda la información que sea relevante en ese punto, haciendo que se comprenda mejor lo llevado a cabo en el trabajo.
La planificación que se ha llevado a cabo en este proyecto es la siguiente:

1. **Fase de formación.** En un primer lugar se comenzó con la parte de estudio y aprendizaje sobre el tema en el cual se fundamenta este proyecto, consiguiendo así saber mejor la dirección por la cual debería seguir nuestro proyecto para conseguir los objetivos marcados.
2. **Fase de investigación.** Se continuó con la búsqueda de información y de proyectos que fuesen de interés, pudiendo ser estos últimos el punto de partida del nuestro.
3. **Fase de desarrollo.** Tras la fase más puramente de análisis se comenzó la parte de plasmar las ideas finales resultantes de las fases anteriores. Se creó un programa que ayudase a mostrarlas de una forma más clara y sencilla.
4. **Fase de verificación y resultados.** Una vez se terminó la fase de desarrollo, dio comienzo la fase encargada de la comprobación de que todo lo planteado en los puntos anteriores fuese lo esperado. Para ello se muestran diferentes salidas generadas por el proyecto.
5. **Fase de documentación.** Esta fase es la encargada de recopilación y organización de toda la información relacionada con el proyecto poniéndola en un punto común, en la memoria, de una forma tal que se comprenda todo lo relevante sobre el proyecto. Debido a esto, esta fase aparece de forma paralela en todas las anteriormente citadas.

Una vez finalizadas todas ellas dan como resultado nuestro proyecto final, el cual explicaremos con mayor detenimiento las diferentes secciones en las cuales está dividida la memoria.

2. Estado del arte

En esta sección de la memoria expondremos cómo se encuentra el tema del cual se basa este trabajo en la actualidad para saber de dónde partimos y hacia dónde nos dirigimos.

En un primer lugar, haremos una introducción a las herramientas Honeypots para saber qué son, para qué se utilizan, su clasificación, las diferentes disposiciones que se pueden dar de los Honeypots y las principales ventajas e inconvenientes que éstos tienen.

Posteriormente, mostraremos una clasificación de los diferentes ataques que pueden sufrir los sistemas informáticos a igual que los Honeypots, como se verá en la primera sección, centrándonos principalmente en aquellos ataques de SSH e inyección SQL.

Seguidamente, presentaremos los sistemas de detección de intrusos (IDS), su definición, su utilidad, sus tipos, así como sus tipos en función de la reacción frente a los ataques que reciban.

A continuación, explicaremos la utilidad de un componente de un framework integrado en el kernel de Linux llamado Iptables, que permite la gestión de los paquetes de red que están presentes en cualquier sistema informático.

Finalmente, mostraremos por qué se ha elegido esta realización del proyecto frente a otras ya existentes.

2.1 Honeypots

Para la detección de intrusiones en los sistemas informáticos una de las herramientas que se utilizan con más frecuencia es la herramienta Honeypot.

Como bien indica su nombre, se encarga de ser "el tarro de miel" para que los atacantes vayan hacia él, ya que estará más desprotegido contra los ataques y tendrá una (falsa) información más interesante para los atacantes, provocando que éstos dejen de lado los otros componentes de la red que estén conectados con el Honeypot.

Veamos una definición más formal:

> ***Honeypot:*** *Es todo aquel sistema que sirve de señuelo diseñado para atraer a un atacante potencial y quitar su atención de los sistemas críticos.*

Los Honeypots están diseñados principalmente para: desviar a un atacante del acceso al sistema crítico, coleccionar información sobre la actividad del atacante y alentar al atacante a que permanezca en el sistema el tiempo suficiente para recabar información sobre el ataque y tomar medidas al respecto.

Por tanto, después de ver todo esto llegamos a la conclusión de que los Honeypots no son sistemas que protegen o ayudan a aumentar la protección de un sistema contra los ataques, sino que sirven para que sean atacados éstos en vez de los sistemas principales.

2.1.1 Clasificación de los Honeypots

Los Honeypots se pueden clasificar de dos formas:

Según su uso:

- **Honeypots para la producción:** Son fáciles de usar, capturan una información limitada y son utilizados principalmente por compañías o corporaciones. Este tipo de

Honeypot está situado dentro de la red de producción con otros servidores de producción y van recopilando información durante todo el tiempo en que son atacados.

- **Honeypots para la investigación:** Este tipo de Honeypot se ejecuta para recopilar información sobre los patrones o tácticas que utilizan los atacantes para acceder a los sistemas informáticos para posteriormente, con la información recogida, aprender a protegerse contra esos ataques.

Según su nivel de interacción:

- **Honeypots de baja interacción:** Este tipo de Honeypot sólo simula los servicios que son solicitados frecuentemente por los atacantes. Por esta razón, la gran ventaja de este tipo de Honeypot es su simplicidad y el relativamente bajo consumo de los recursos del sistema.

  La desventaja de este tipo de Honeypot es la limitación de la simulación, ya que al simular los servicios que son más frecuentes por los atacantes, un atacante puede llegar a darse cuenta de que el sistema que ataca es un Honeypot.

  Un ejemplo de este tipo de Honeypot es: Honeyd.

- **Honeypot de alta interacción:** Este tipo de Honeypot imitan las actividades de los sistemas de producción en los cuales están situados.

  Por esta razón, un atacante tiene una cantidad suficientemente grande de servicios en los que puede perder el tiempo y dejar el sistema principal libre del ataque. Y la gran ventaja de este tipo es la ingente variedad y cantidad de información que se recoge, ya que el Honeypot es igual que un sistema real, y es más difícil de que el atacante detecte este hecho.

  El inconveniente de este tipo de Honeypot es su gran coste de mantenimiento, debido a que simulan un sistema real, y si el atacante se da cuenta de que es un Honeypot podría llegar a utilizarlo para su beneficio.

  Un ejemplo de este tipo de Honeypot es: Honeynet.

- **Honeypot híbrido:** Este tipo de Honeypot combina las herramientas de los dos tipos mencionados anteriormente (baja y alta interacción) para tener los beneficios y mitigar los inconvenientes de ambos tipos de Honeypot.

2.1.2 Ubicación de Honeypots

Los Honeypots pueden ser instalados en diferentes lugares de la red, todo dependerá del tipo de información que estemos interesados en conseguir y el nivel de riesgo que queramos llegar a alcanzar para recopilar la mayor cantidad de datos posible.

Por tanto, se pueden diferenciar tres ubicaciones posibles para un Honeypot:

- **Honeypot antes del Firewall:** El Honeypot se sitúa antes del Firewall del sistema para un mejor seguimiento de los intentos de conexión a la red mediante direcciones IP que no se están utilizando.

  Colocado el Honeypot de esta forma no aumenta el riesgo de que los ataques se produzcan en el interior del sistema, por lo que el peligro de comprometer el sistema

se evita al igual que la activación de las alarmas que pudieran saltar en el Firewall y en los sensores del IDS (Sistema de detección de intrusión), ya que todos los ataques irán hacia el Honeypot.

Un inconveniente es que perdemos todos los ataques que se dan en el interior del sistema.

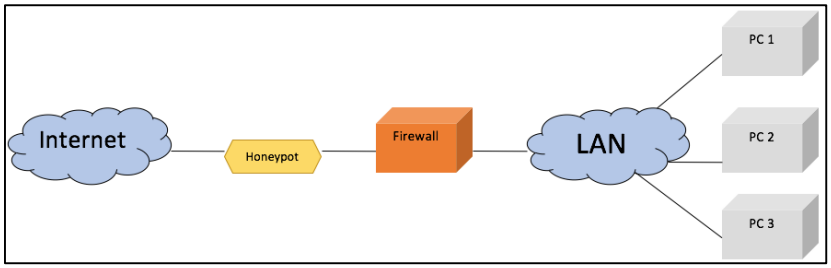


Ilustración 1. Esquema Honeypot antes del Firewall.

- **Honeypot detrás del Firewall:** El Honeypot se sitúa detrás del Firewall del sistema al mismo nivel de la red interna.

  Las grandes ventajas de esta ubicación del Honeypot son: puede capturar ataques que se hagan en la red interna; y puede detectar una desconfiguración del Firewall del sistema para que permita el acceso de tráfico directamente desde Internet a la red interna.

  Las desventajas de esta ubicación son: el sistema está comprometido frente a los ataques, ya que el Honeypot se encuentra al mismo nivel que los elementos de la red interna; los ataques que pudiese frenar el Firewall podrían acceder al sistema ya que formaría parte del tráfico permitido para el Honeypot; y se debería configurar el Firewall de forma que permita el tráfico hacia el Honeypot, incrementando el nivel de riesgo de la red interna hacia los ataques.

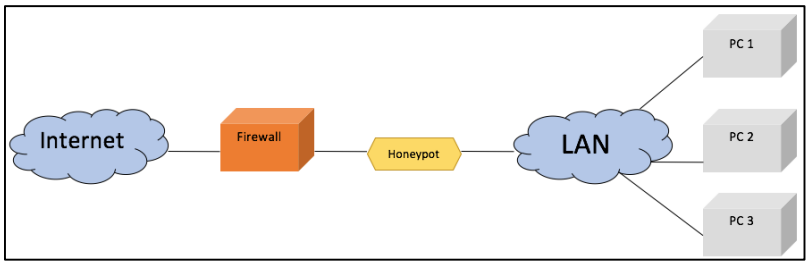


Ilustración 2. Esquema Honeypot detrás Firewall.

- **Honeypot en zona desmilitarizada (DMZ):** Al igual que el anterior tipo de ubicación de Honeypot, se sitúa detrás del Firewall del sistema, pero en este caso se coloca junto a los servidores de producción, lo que da mayor seguridad a la red interna.

  Esta ubicación nos permite detectar tanto ataques internos como externos de la red donde está situado, tras la realización de una reconfiguración en el Firewall. Además se eliminan las alertas que pudiesen producirse por un IDS y el peligro de recibir

ataques en la red interna, ya que la ubicación de este Honeypots no está en contacto con ella.

La desventaja de esta ubicación es la pérdida del número de ataques internos, debido a que no comparte el mismo segmento que la red interna, ocasionando que los ataques no se dirijan hacia el Honeypot a menos que lo hagan explícitamente.

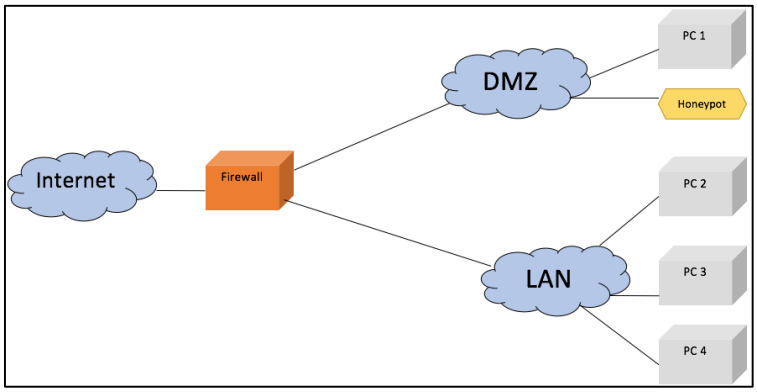


Ilustración 3. Esquema DMZ.

2.1.3 Honeynet

Simplificando podría decirse que una Honeynet es un red de Honeypots, pero esta afirmación no es correcta en su totalidad. Honeynet es un proyecto creado alrededor de 1999, cuyo fundador Lance Spitzner definió Honeynet de la siguiente forma:

> ***Honeynet:*** *Red de Honeypots de alta interacción que simula una red de producción configurada de tal forma que toda actividad en ella está monitorizada, grabada y de algún modo discretamente regulada.*

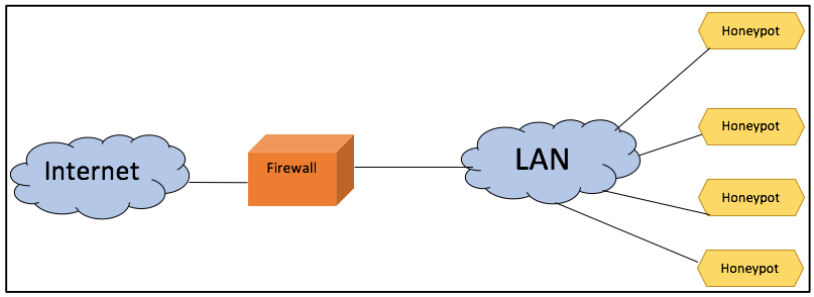


Ilustración 4. Esquema Honeynet.

Por tanto, una Honeynet es utilizada, fundamentalmente, para la investigación y la obtención de información lo más detallada posible de los ataques que se producen en una red.

Como una Honeynet está compuesta por varios Honeypots, los cuales recibirán ingentes cantidades de ataques, ésta deberá situarse en un lugar que no comprometa a los otros componentes de la red, eliminando así el riesgo de que éstos sufran un ataque.

De esta manera, en nuestro trabajo se utilizará una Honeynet para recaudar un mayor número de información sobre los ataques sufridos por los Honeypots instalados. Se dispondrán diferentes tipos de Honeypots en distintas direcciones IP en la red de la UAM situadas fuera

de la red normal, para no comprometer los demás sistemas informáticos conectados a ella. Se enviará la información a un servidor para disponer de ella de forma centralizada.

2.1.4 Ventajas y desventajas de los Honeypots

Los Honeypots presentan las siguientes ventajas:

- Recogen una cantidad pequeña de datos pero de gran utilidad y valor para su estudio.
- Generalmente son fáciles de diseñar, implementar y su uso ayuda a mejorar las condiciones de seguridad de cualquier sistema.
- El número de falsos positivos y de falsos negativos. Los Honeypots son sistemas a los que ningún otro componente de la red debería acceder, centrándose únicamente en los ataques provenientes del exterior de la red.
- Los recursos que necesita son mínimos. No consumen ancho de banda, ni memoria o CPU extra.

Por otro lado se pueden observar los siguientes inconvenientes de utilizar los Honeypots:

- Son sistemas totalmente pasivos, es decir, que si no reciben ningún ataque resultan inútiles.
- Introducen riesgo en los entornos en los que se utilizan puesto que son herramientas configuradas para ser atacadas. Para mitigar este riesgo se puede utilizar la ubicación DMZ.
- No añaden seguridad al sistema o red en los cuales están instalados. Ayudan a recoger información con el fin de analizarla y después tomar las medidas necesarias para que los ataques no se vuelvan a producir.

2.2 Tipos de ataque

Antes de empezar la clasificación de los tipos de ataque informático en necesario introducir la definición de seguridad informática.

> ***Seguridad informática:*** *Protección otorgada a un sistema de información automatizado con el fin de obtener los objetivos aplicables a la preservación de la integridad, disponibilidad y confidencialidad de la información de los recursos del sistema.*

En esta definición se muestran tres importantes conceptos: integridad, disponibilidad y confidencialidad. Estos tres conceptos son claves para clasificar los ataques informáticos. Por tanto, deberemos introducir la definición de cada uno para conocer el objetivo del ataque.

- **Integridad:** Existen dos tipos de integridad: la integridad de datos y la integridad del sistema.
    - **Integridad de datos:** Asegura que tanto la información como los programas son modificados únicamente de forma específica y autorizada.

- **Integridad del sistema:** Asegura que el sistema realiza su función de manera irreprochable, libre de una manipulación deliberada o inadvertida no autorizada del sistema.

- **Disponibilidad:** Asegura que el sistema funciona con fluidez y el servicio no está denegado para usuarios autorizados

- **Confidencialidad:** Existen dos tipos de confidencialidad: la confidencialidad de datos y privacidad:
  - **Confidencialidad de datos:** Asegura que información privada o confidencial no está disponible para individuos no autorizados.
  - **Privacidad:** Asegura que los individuos controlan qué información relacionada con ellos mismos puede ser almacenada o recogida y por y para quién esa información puede ser accesible.

Atendiendo a la definiciones presentadas anteriormente, los ataques informáticos buscarán las debilidades del sistema para dañar la integridad, la disponibilidad o la confidencialidad del mismo.

Esta forma de actuar ha generado un gran número de taxonomías de tipos de ataques informáticos, por ejemplo, las dos siguientes:

- **Pasivos o activos:** Dependiendo de si afectan o no a los recursos del sistema, respectivamente.

- **Internos o externos:** Dependiendo de si el ataque se realizada desde dentro o fuera del sistema, respectivamente.

Sin embargo la taxonomía más extendida es la que se presenta a continuación:

- **Denegación de servicio (DoS):** Acción que impida o menoscabe el uso autorizado de redes, sistemas o aplicaciones mediante el agotamiento de recursos tales como la CPU, memoria, el ancho de banda y el espacio en disco.
  Atendiendo a estas características se pueden diferenciar varios tipos de DoS, dependiendo de los recursos que se pretenda atacar: al ancho de banda, a los recursos del sistema y a los recursos de aplicaciones.

- **Indagación o Exploración (Probing):** Este tipo de ataque se encarga de escanear una computadora o una red para determinar las debilidades o vulnerabilidades que pueden ser explotadas más tarde para comprometer el sistema.

- **Remote to Local (R2L):** En este tipo de ataques se intenta explotar las vulnerabilidades del sistema con el fin de controlar de forma remota la red de la computadora atacada simulando ser un usuario local autorizado.

- **User to Root (U2R):** En este tipo de ataques se intenta obtener los derechos de acceso de un host normal para conseguir posteriormente el acceso root (súper usuario) en el sistema.

Los dos últimos puntos de esta clasificación serán la base de estudio de nuestro trabajo, debido a que son los más frecuentes en los sistemas. Los atacantes quieren encontrar los

puntos débiles de los sistemas para poder acceder a ellos ya sea intentado encontrar la pareja de valores usuario-contraseña como utilizando inyección SQL. Por ello, todas la reglas que se estarán orientadas a aumentar las barreras que existen frente a este tipo de ataques (intento de acceso con usuario-contraseña e insertar código SQL).

2.3 Sistema de detección de intrusión (IDS)

Los sistemas de detección de intrusión son dispositivos o aplicaciones software que monitorizan las tareas de una red o un sistema ante las actividades malintencionadas o violaciones de política (escaneo de puertos, paquetes malformados, ataques por fuerza bruta, etc.) y produce informes a una estación de mantenimiento.

Los IDS analizan tanto el tipo, el contenido y el comportamiento del tráfico que atraviesa la red o el sistema donde estén instalado. Por esta razón normalmente un IDS se integra junto con un Firewall para aumentar la seguridad de la red, ya que el primero se encarga de la detección de ataques y el segundo de su bloqueo.

2.3.1 Tipos de IDS

Los tipos de IDS que se pueden diferenciar son:

- **Sistema de detección de intrusión en red (NIDS):** Este tipo de IDS se sitúa en un punto o unos puntos estratégicos dentro de la red para monitorizar el tráfico de todos sus dispositivos, tanto el de entrada como el de salida. Éste realiza un análisis del tráfico de toda la red y comprueba en base a la librería de ataques conocidos, si los paquetes de entrada a la red son potenciales ataques. En caso afirmativo lo reporta al administrador.

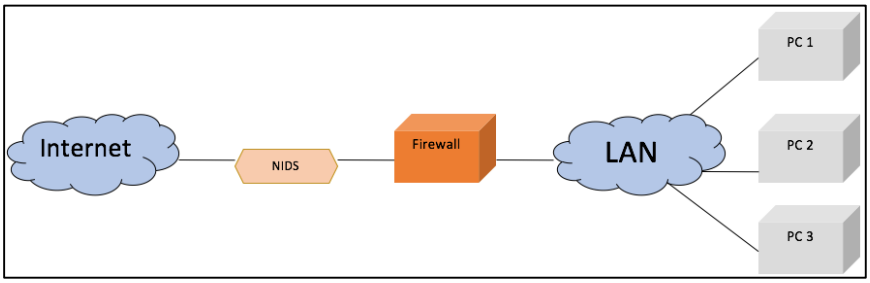


Ilustración 5. Esquema NIDS.

- **Sistema de detección de intrusión en host (HIDS):** Este tipo de IDS se sitúa solamente en un host o un dispositivo de la red, donde monitoriza los paquetes que entran y salen del sistema alertando al usuario o al administrador cuando detecte algo sospechoso.

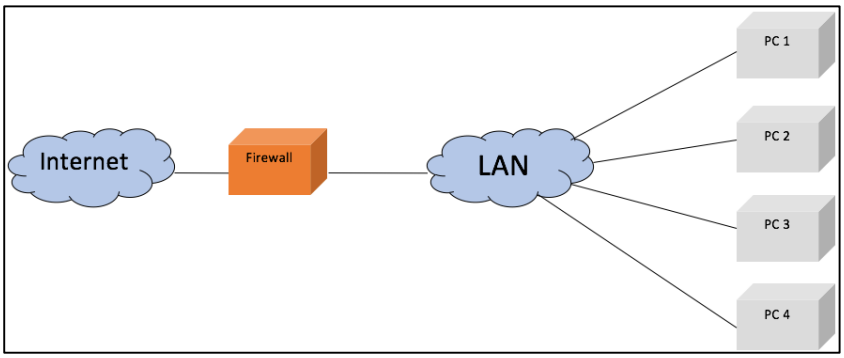


Ilustración 6. Esquema HIDS.

2.3.2 Tipos de sistemas

Hay dos tipos de sistemas atendiendo al comportamiento del IDS:

- **Pasivos:** El IDS detecta un ataque potencial, guarda información sobre ese ataque y manda una señal de alerta.

- **Reactivos:** El IDS responde al ataque reprogramando el Firewall para que bloquee las entradas provenientes de la red atacante. También son conocidos como sistemas de prevención de intrusión (IPS).

Nuestra investigación se basa en los IDS. Concretamente un IDS adaptativo, es decir, un IDS que se adaptará a los ataques que recibe para añadir nuevas reglas a su librería de ataques comunes. Siendo unos de los objetivos mejorar la seguridad de los sistemas informáticos.

Para ello, se utilizarán Honeypots, que recibirán los ataques y recabarán información sobre los mismos. Dicha información será recogida para ser estudiada creando y añadiendo nuevas reglas a nuestro sistema informático, mejorando, así, su seguridad.

2.4 Netfilter/Iptables

Netfilter [10] es un framework proporcionado por el kernel de Linux desde su versión 2.4.x, como sucesor de ipchains de Linux 2.2.x e ipfwandn de Linux 2.0.x que permite realizar varias operaciones con los paquetes que recibe la red, ayudando a crear un manejador de paquetes según las necesidades del usuario. Provisiona al sistema operativo de funciones y operaciones para el filtro de paquetes, la traducción de direcciones de red y traducción de puertos, lo que a su vez, ayuda a poder dirigir los paquetes recibidos por el sistema a través de una red de sistemas informáticos o para bloquear el recibimiento de paquetes de direcciones IP.

Iptables [10][11] es un componente de Netfilter que se ofrece al usuario mediante la línea de comandos se permite configurar las tablas encargadas del filtrado de los paquetes que recibe el sistema informático así como las cadenas y reglas que almacena cada una de la tablas. Debido a la importa que supone manejar la información de los paquetes que recibe el usuario, ésta sólo puede ser modificada por el súper usuario del sistema.

Iptables dispone de tres componentes para el manejo de paquetes de red:

- **Reglas:** Son una serie de condiciones introducidas por el súper usuario para controlar los paquetes de red. Sirven para indicar qué debe hacer el sistema con los paquetes recibidos.
- **Cadenas:** Es una lista ordenada de reglas.

- **Tablas:** Es un conjunto de cadenas. Cada tabla está asociada a cada tipo de procesamiento de paquetes, por tanto, las cadenas de una tabla tendrán un factor común a la hora de cómo manejar los paquetes de red.

2.4.1 Procesamiento de un paquete por Iptables

Cuando un sistema informático envía o recibe un paquete de red éste es procesado por al menos una cadena del total de las que están almacenadas en las tablas. Tras determinar qué tipo de procesamiento tiene el paquete se selecciona la tabla pertinente, se van recorriendo las cadenas guardadas en ella y comparando las condiciones de cada regla que hay en cada cadena hasta que los términos del paquete y la regla sean coincidentes, en este caso se realiza la acción dictada por la regla. En caso contrario, es decir, si no hay ninguna regla que defina lo que se debe hacer con el paquete, se realizará la política por defecto que indique la tabla, ya sea desechar el paquete o enviarlo a otra de las tablas o cadenas.

2.4.2 Tablas Iptables

Iptables dispone de tres tablas ya creadas desde en inicio, las cuales tienen ciertas cadenas predefinidas. El súper usuario podrá crear nuevas tablas en el sistema por medio de paquetes de extensión que lo permiten así como crear, modificar y/o eliminar cadenas de cada una de las tablas existentes en el sistema. Al comienzo, las cadenas que tienen cada una de las tablas se encuentran vacías, por lo que todo paquete que se recibe no pasa por ningún filtro y llega íntegro. En este caso la seguridad presente en el sistema es baja o incluso nula. Por ello, la motivación de este proyecto es crear un conjunto de reglas que formarán cadenas y se incorporarán en la tabla que sea oportuna, para la mejora de la seguridad. Con estas reglas y cadenas se dificultará el acceso de paquetes que puedan ser dañinos para el sistema.

A continuación mostramos las tablas creadas inicialmente por Iptables así como las tareas de las que se encargan y las diferentes cadenas que las integran:

- **Filter table (tabla de filtros):** Es la encargada del filtrado de los paquetes de red, es decir, permite o bloquea el recorrido del paquete por el sistema. Todos los paquetes de red, entrantes y salientes, pasan por esta tabla. Está compuesta por las siguientes cadenas:
    - **INPUT (LOCAL_INPUT):** Todos los paquetes que llegan al sistema pasan por esta cadena.
    - **OUTPUT (LOCAL_OUTPUT):** Todos los paquetes que se envían desde el sistema pasan por esta cadena.
    - **FORWARD:** Se encarga del redireccionamiento de los paquetes.

- **NAT table:** Se encarga de la configuración de las reglas relacionadas con la traducción de las direcciones de red. Permite reescribir la traducción de las direcciones de red, incluyendo los puertos asociados a las mismas. Solamente pasa por esta tabla el primer paquete de cada conexión que haya en el sistema, ya que se guarda la reescritura para los siguientes paquetes. Está compuesta por las siguientes cadenas:
    - **PREROUTING:** Los paquetes recibidos por el sistema pasan en primer lugar por ella para hacer la redirección de destino (DNAT) en caso de ser necesario.

- **POSTROUTING:** Los paquetes que salen del sistema pasan por esta tabla para modificar la dirección de origen (SNAT).
- **OUTPUT:** Permite generar un DNAT pero sólo con aquellos paquetes generados de forma local.
- **INPUT:** Permite generar un DNAT pero sólo con aquellos paquetes que tienen como destino el propio sistema.

- **Mangle table:** Se encarga del ajuste de las opciones (parte final del datagrama) de todos los paquetes de red del sistema, tanto entrantes como salientes. Como por ella pasan todos los paquetes del sistema tiene que estar integrarda por todas las cadenas predefinidas que hemos vistos en los puntos anteriores:
  - **PREROUTING:** Todos aquellos paquetes que consiguen acceder al sistema pasan por esta cadena antes de que se tome cualquier decisión por parte de las cadenas de INPUT o FORWARD.
  - **INPUT:** Todos los paquetes que llegan al sistema pasan por esta cadena.
  - **OUTPUT:** Todos los paquetes que se envían desde el sistema pasan por esta cadena.
  - **FORWARD:** Aquellos paquetes que sólo pasan a través del sistema pasan por ella.

Adjuntamos una ilustración para mostrar lo que hemos explicado en los puntos anteriores.

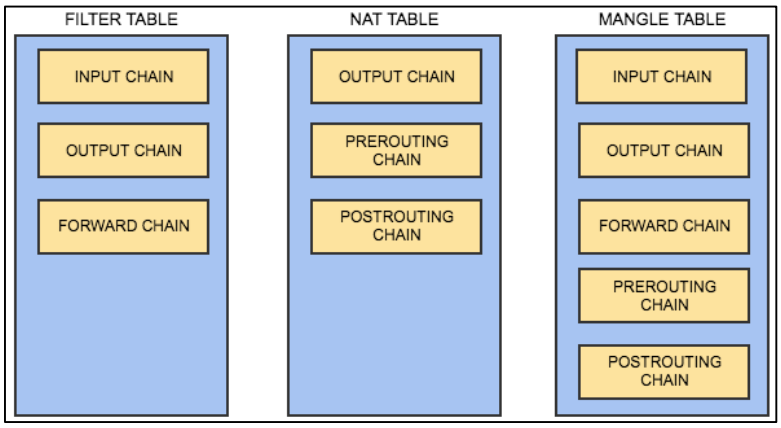


Ilustración 7. Tablas predeterminadas Iptables

Adicionalmente, el súper usuario podrá incorporar todas aquellas cadenas que le sean necesarias o crea oportunas de incluir en cada una de las tablas mencionadas con anterioridad.

2.4.3 Destinos de las reglas de Iptables

Las diferentes reglas que pueda crear un usuario tienen la posibilidad de ser dirigidas a los diferentes destinos que vienen por defecto en Iptables: ACCEPT, DROP, QUEUE o RETURN, que explicaremos más adelante. Cuando el destino es una regla creada por el usuario, el paquete es redirigido a esa cadena para ser procesado. Si tras el procesamiento el paquete llega hasta el final de la cadena sin que ninguna de las reglas de la cadena haya

actuado sobre éste, el paquete es enviado nuevamente al punto de inicio, es decir, a la primera cadena procesadora. Esta funcionalidad de anidamiento de cadenas permite que la criba de los paquetes dañinos que puedan llegar a nuestro sistema sea menor, incluso una posibilidad casi nulo. Por este motivo, se ha elegido utilizar esta herramienta en nuestro proyecto.

Descripción de los diferentes destinos:

- **ACCEPT:** Hace que el paquete sea aceptado en el sistema. Dependiendo de la cadena en la cual se encuentre el paquete tendrá un significado diferente: si es la cadena INPUT será recibido por el sistema, si es en la cadena OUTPUT se le permitirá abandonar el sistema y si es la cadena FORWARD se le permitirá ser redireccionado en el sistema.
- **DROP:** Hace que el paquete sea descartado y no siga ningún otro procesamiento, siendo eliminado del sistema.
- **QUEUE:** Hace que el paquete sea enviado a una cola dentro del espacio de usuario. Si no hay ninguna acción por parte de algún programa y/o aplicación sobre esa cola, este destino será equivalente al destino DROP.
- **RETURN:** Hace que el paquete deje de ser procesado por la cadena si este ha hecho que alguna de las reglas haya actuado sobre ella y tenga este tipo de destino. Si la cadena en la cual se ha dado la acción sobre el paquete es una subcadena de otra por debajo de una principal, el paquete será enviado a esa cadena principal y seguirá el procesamiento de ésta. En caso contrario, si se trata de una de las cadenas principales, se realizará sobre el paquete la política predeterminada asociada a esta cadena principal.

Además, existen otros tipos de destino que integran extensiones de los anteriores mencionados. Citamos los más utilizados: REJECT, LOG, ULOG, DNAT, SNAT y MASQUERADE.

En la ilustración 8se muestra el flujo de procesamiento de un paquete. Cabe destacar que la tabla nombrada como ‘raw’ es la tabla que procesa el paquete antes de que se establezca la conexión entre el sistema receptor y emisor de paquetes de red.

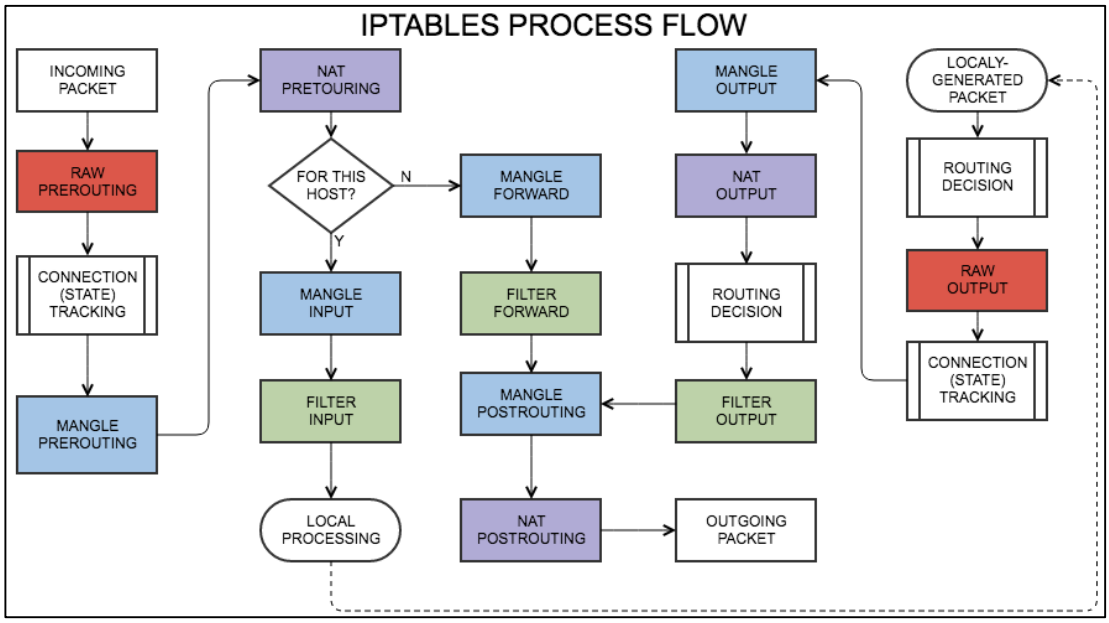


Ilustración 8. Esquema procesamiento de un paquete.

2.5 Trabajos relacionados con la propuesta

Tras investigar sobre el tema en que se basa este proyecto, encontramos una serie de artículos que exponen la creación de sistemas adaptativos mediante la información que almacenan algunos IDS instalados en los sistemas informáticos [12][13][14], tales como SNORT [16], y algunos que adaptaban Honeypots de media-alta interacción realizando acciones utilizando métodos de la Teoría de Juegos [18] con la información de los Honeypots [15][17].

A continuación mostraremos algunos de los trabajos relacionados con la propuesta de nuestro proyecto, resaltando los aspectos más interesantes y remarcando aquello que será innovador en nuestra propuesta.

2.5.1 Introducción a la Teoría de Juegos

Comenzamos explicando los conceptos básicos de la Teoría de Juegos, ya que algunos de los trabajos que expondremos más adelante la emplean para la toma de decisiones en su sistema frente a los atacantes.

La Teoría de Juegos es una rama de la matemática aplicada que estudia modelos matemáticos que analizan la interacción en estructuras formalizadas de incentivos (juegos) generando procesos de decisión [19]. A través de la observación de los resultados en los juegos se generan procesos de decisión para futuros sucesos similares. Los Honeypots se pueden tratar de esta manera, debido a que los atacantes que acceden a ellos navegan de diferentes formas (simulan sucesos diferentes), dejando un rastro tras ellos. Esto permite poder tomar decisiones frente a futuros ataques, suponiendo que los comportamientos futuros debe ser similares a los comportamientos ya realizados por otros atacantes.

La forma más conveniente para la representación del comportamiento de los Honeypots se identifica con la forma extensiva utilizada en la Teoría de Juegos. Este tipo de representación muestra un esquema en forma de árbol, donde cada vértice denota un punto en el cual un jugador tomará una decisión.

A continuación se muestra un esquema de juegos en Forma Extensiva:

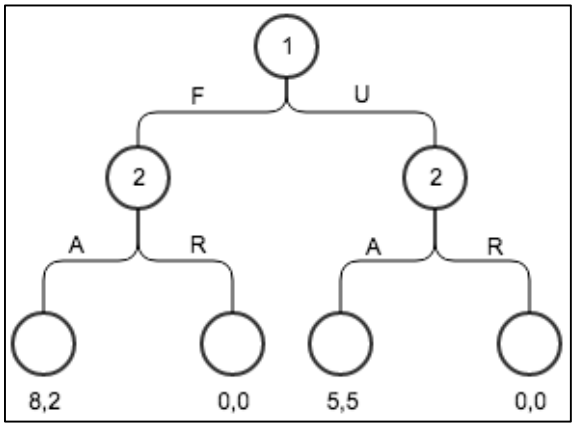


Ilustración 9. Esquema juego en forma extensiva.

En el esquema anterior, en primer lugar, el jugador 1 elige entre dos caminos (F y U), cuando se decide por uno de ellos es el turno del jugador 2 que deberá elegir entre otros dos caminos diferentes a los dos del jugador 1 (A y R). Cuando se llega a la zona más baja del árbol se reparten puntos entre ambos jugadores, dando lugar, en este caso, a que el jugador 2 nunca gane ya que la repartición entre ellos es siempre a favor de jugador 1 o equitativa entre los dos jugadores (8>2, 0=0, 5=5, 0=0).

2.5.2 La Teoría de Juegos en los Honeypots

En este apartado mostramos una visión general de dos trabajos relacionados con la Teoría de Juegos en los Honeypots: "Self-Adaptive Honeypots Coercing and Assessing Attacker Behaviour" [17] y "Adaptative and Selfconfigurable Honeypots" [15]. Ambos utilizan la Teoría de Juegos en Forma Extensiva donde la elección de los caminos son probabilidades y la suma de todos los salientes del vértice es 1. Los vértices son diferentes comandos que puede ejecutar el atacante.

En ambos trabajos se utiliza el siguiente esquema.

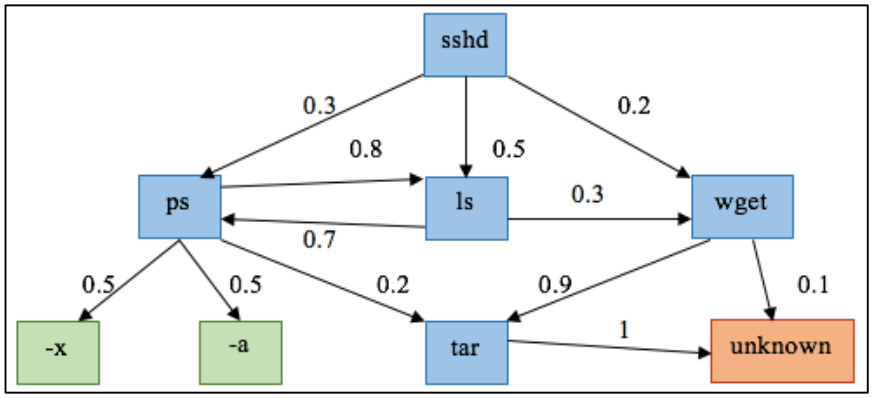


Ilustración 10. Esquema trabajos Teoría de Juegos.

Como podemos observar el esquema comienza con el vértice SSHD (daemon de ssh), utilizado para estudiar las conexiones por ssh al sistema. Este árbol identifica cuando un atacante ya ha entrado en el sistema, y los posteriores comandos que realiza este atacante una vez ha accedido. El paso de un nodo a otro es una probabilidad entre 0 y 1, verificando que la suma de todos los caminos salientes de cada nodo es igual a 1.

Tras el esquema inicial los dos trabajos difieren en los esquemas de tratamiento de los atacantes. El primero utiliza la forma de árbol mientras que el segundo un tabla con los valores de probabilidad de cambio.

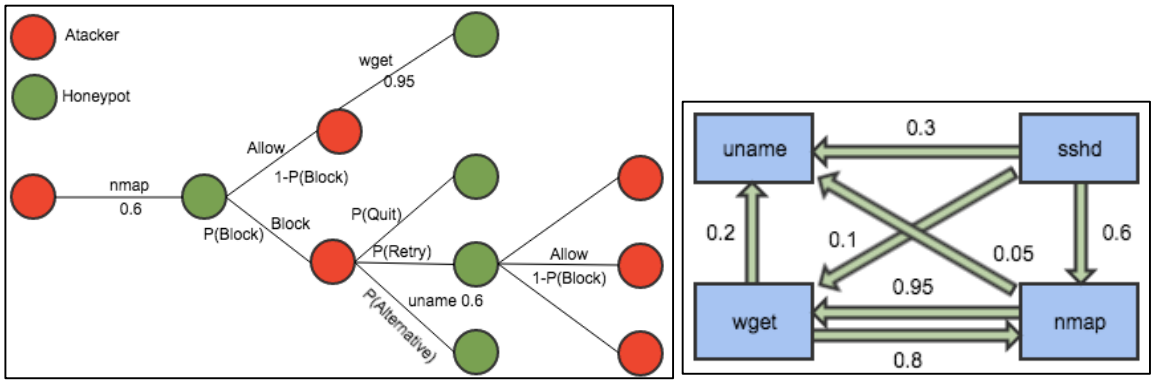


Ilustración 11. Esquema juego extraído de [15].

En la ilustración 11 podemos apreciar la interacción del atacante y el Honeypot según van tomando decisiones cada uno de ellos, el primero ejecutando comandos dentro del sistema y el segundo permitiéndole o denegándole esa acción.

	sshd	*bash*	*uname*	*ps*	*wget*
sshd	ε/4	1(1-ε)	ε/4	ε/4	ε/4
bash	ε/3	ε/3	2(1-ε)/3	(1-ε)/3	ε/3
uname	ε/4	ε/4	ε/4	ε/4	1(1-ε)
ps	ε/4	1(1-ε)	ε/4	ε/4	ε/4
wget	ε/4	ε/4	1(1-ε)	ε/4	ε/4

Tabla 1. Matriz de transición de probabilidades extraída [17].

En la Tabla 1 asociada al segundo trabajo podemos apreciar que se utiliza un valor épsilon para suavizar las probabilidades. Cuando los valores son mayores que 1 (valor no valido al tratarse de probabilidades) se les multiplica por (1 - épsilon) para que la suma de todos los elementos de cada fila sumen 1.

Los trabajos confluyen en que los Honeypots "capturan" a los atacantes para que se mantengan en el sistema el máximo tiempo posible y así poder saber más sobre ellos: cómo atacan y las maniobras que hacen cuando se le impide que sigan un camino, que, seguramente, tengan predeterminado. Esto genera mucha información sobre cómo navegan los atacantes por el sistema por tanto, la recopilación de la información se vuelve proporcional al tiempo que el atacante pasa en el sistema.

Esta investigación expuesta en los trabajos resulta muy interesante puesto que permite saber más acerca de los atacantes en el caso en el que se les ponen distintos escoyos en su camino a través del sistema. Sin embargo, los resultados de la investigación dependen del tiempo trascurrido entre la entrada y la salida del atacante del sistema, lo que implica un gran inconveniente. Por esta razón desestimamos enfocar nuestro trabajo en esta dirección.

2.5.3 Trabajos utilizando la información de los IDS

En este apartado mostramos una visión general de algunos de los siguientes tres trabajos relacionados con la utilización de la información de los IDS: "Adaptive Real Time Intrusion Detection Systems" [12], "Real-Time Adaptive Security" [13] y "Design of Adaptive IDS with Regulated Retraining Approach" [14].

Todos ellos utilizan la información que almacenan los IDS creados con anterioridad para realizar alguna acción frente a los atacantes del sistema, dando como resultado un IDS adaptativo. Esto tiene la ventaja principal de que están creados de tal forma que la información almacenada ya se encuentra procesada. El inconveniente de estos IDS no presentan vulnerabilidades frente a los atacantes, lo que supone que la cantidad de información que pueden albergar es menor que si se utilizase un Honeypot para recopilar dicha información.

Por esta razón, decidimos utilizar la información que nos proporcionan los Honeypots para poder tomar decisiones frente a los atacantes, consiguiendo un mayor volumen de datos frente al obtenido si utilizásemos un IDS ya configurado.

Durante la búsqueda de trabajos relacionados con la propuesta se encontró una investigación en la línea de los anteriores, es decir, que proponía la creación de un IDS adaptativo, que con novedad utilizaba un Honeypot "pequeño", para recopilar información de los atacantes, con objetivo de mejorar la seguridad del IDS [20]. Incorporando reglas creadas manualmente tras el análisis de la información del Honeypot. Este trabajo se asemeja a la propuesta de

nuestra investigación, ya que utiliza Honeypots para ayudar a mejorar la seguridad de nuestro sistema frente a los ataques que este sufre. La novedad de nuestra propuesta radica en la utilización de un mayor número de Honeypots con diferentes singularidades para recopilar información que permita la generación de reglas automáticas.

En conclusión, nuestro objetivo es detectar potenciales intrusos en trampas creadas, Honeypots, recopilando información relevante sobre ellos para su posterior procesamiento. Por esta razón, se ha decidido crear un pseudo-IDS, incorporando reglas creadas automáticamente, usando la información recogida por los Honeypots. Las reglas creadas automáticamente se añadirán en el Iptables del sistema para bloquear a los atacante a nivel de red, otra situación novedosa en nuestra investigación.

3. Análisis

En esta sección expondremos el análisis realizado para la elaboración de nuestra infraestructura de creación de reglas automáticas.

En primer lugar, se decidió crear una red propia de Honeypots conformada por diferentes tipos de Honeypots. Esta decisión recae en la posibilidad obtener una mayor variedad de información a analizar debido a los diversos tipos de ataques que sufren los sistemas informáticos. Estos Honeypots deberán situarse en distintas direcciones IP, lo que nos ayudará a tener un mayor número de focos de atracción de los atacantes. Para tener un mejor manejo, acceso y seguridad de la información se decidió centralizar el almacenamiento de toda esta información en un servidor externo. Resulta importante destacar que todo este proceso se llevó a cabo en colaboración con el proyecto de Diego Jurado Pallares [21].

Por último, se determinó la creación de una infraestructura implementada en lenguaje Python, que proporciona las facilidades y funcionalidades necesarias para el manejo de grandes cantidades de información, así como para el procesamiento de la información almacenada en la base de datos y la posterior creación de las reglas a implantar.

En la Ilustración 12 se muestra el esquema ideado para el desarrollo del programa. Se observa, que la información de los ataques de los Honeypots se tendrá almacenada en una base de datos que serán descargados por nuestro programa para realizar una serie de operaciones con los mismos, siendo esta acción el procesamiento de la información. Seguidamente, tras la finalización del procesamiento, se creará las reglas oportunas. Todo ello tendrá como consecuencia el aumento de la seguridad del sistema frente a los ataques que pueda sufrir en un futuro.

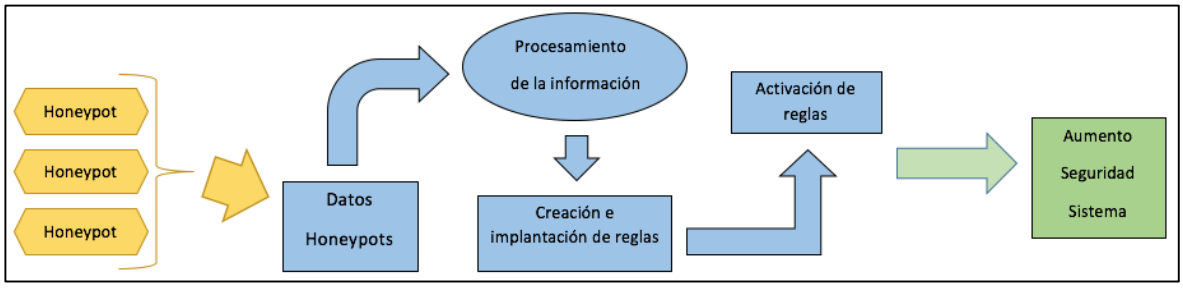


Ilustración 12. Esquema proyecto.

Para conseguir que se siga el flujo marcado tanto por la Ilustración 12 como en la explicación precedente, nuestro programa deberá tener al menos las siguientes funcionalidades:

- Descargar la información relevante de los ataques que reciben los Honeypots almacenada en las tablas de la base de datos MySQL de nuestro servidor.
- Procesar toda la información descargada según una serie de criterios que se hayan marcado y realizar las operaciones pertinentes con ella.
- Creación e implantación de las reglas tras el procesamiento de la información.

3.1 Requisitos funcionales

A continuación se enunciarán los requisitos funcionales de la infraestructura:

RF1. La infraestructura podrá descargar información recopilada de los Honeypots desde la base de datos en MySQL donde se almacena.

RF2. La infraestructura procesará la información descargada según una serie de criterios previamente determinados.

RF3. La infraestructura creará una serie de reglas automáticamente.

RF4. La infraestructura permitirá crear reglas por el usuario para incorporarlas, sólo a nivel de bloqueo de direcciones IP.

RF5. La infraestructura asignará el tipo de regla TMP, a todos aquellos atacantes que no hayan superado el umbral máximo de ataques permitidos por nuestra infraestructura.

RF6. La infraestructura bloqueará a todo aquel atacante que haya intentado acceder más de *n* veces al sistema mediante SSH, siendo *n* un parámetro configurable por el usuario. En este caso, la regla asignada, será la M1.

RF7. La infraestructura bloqueará todo aquel atacante que haya intentado hacer inyección de SQL. En este caso, la regla asignada, será la M1.

RF8. La infraestructura bloqueará a todo aquel atacante que realice un ataque masivo. Se entiende como ataque masivo cuando una dirección IP interactúa al menos una vez con tipos diferentes de Honeypots instalados y que cumplan las condiciones enunciadas en el RF6 y en el RF7. En este caso, la regla asignada, será la M2.

3.2 Requisitos no funcionales

A continuación se enunciarán los requisitos no funcionales de la infraestructura:

RNF1. La infraestructura tendrá una usabilidad sencilla y intuitiva para el usuario.

RNF2. La infraestructura procesará la información almacenada en la base de datos en un tiempo no superior a dos minutos.

RNF3. La infraestructura mostrará la información de aquellos atacantes que más han interactuado con los Honeypots.

RNF4. La infraestructura dará la posibilidad al usuario de bloquear aquellas direcciones de los atacantes que más hayan interactuado con los Honeypots.

4. Diseño

En este apartado se mostrará el diseño de la infraestructura elaborada para alcanzar el objetivo principal de nuestro proyecto. Comenzaremos con el esquema de nuestra red, mostrando los diferentes Honeypots instalados y las conexiones hacia el exterior. A continuación, mostraremos la base de datos y las tablas creadas para el almacenamiento de la información de los ataques recogidos por los Honeypots. Terminaremos describiendo la infraestructura de nuestro proyecto.

4.1 Esquema de la red propia

En la Ilustración 13 mostramos el esquema de nuestra red propia situada en la red de la UAM.

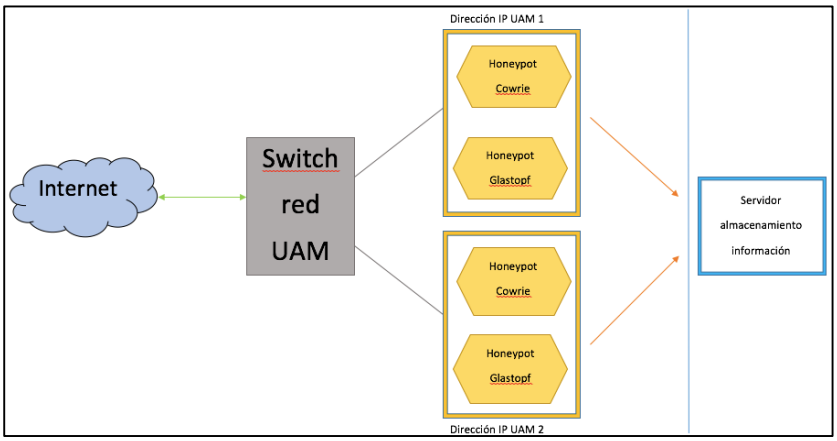


Ilustración 13. Esquema red propia (parte de la red TFG Diego Jurado Pallares).

Como podemos apreciar, el acceso a internet se obtiene a través del Switch de la red de la UAM, el cual asigna dos direcciones IP donde se desplegarán nuestros Honeypots para así ser visibles desde el exterior y puedan así recibir ataques.

Los Honeypot instalados, como podemos apreciar en la imagen, son Cowrie y Glastopf. El primero de ellos nos da información sobre ataques referentes al acceso por ssh, dejando constancia del par usuario-contraseña y de aquellos comandos que ha ejecutado el atacante en el Honeypot. Para que sea posible recibir por inyección de SQL se simula una página web a través del segundo tipo de Honeypot. Esta web contiene un campo de usuario y otro de contraseña, por lo que son éstos susceptibles de ser atacados con el fin de recoger información de la base de datos de la página web.

Se tomó la decisión de sólo tomar dos sensores de la red montada en colaboración con Diego Jurado Pallares, debido a que los tipos de Honeypots instalados en esas dos direcciones eran los que mejor información ofrecían.

4.2 Base de datos

La base de datos creada se sitúa en el servidor externo a la red propia. En ella están las tablas donde se almacena toda la información sobre los ataques que reciben los Honeypots. Estas tablas se crean durante el despliegue de los Honeypots de forma automática ya que hay un fichero en cada Honeypot que realizan esa acción.

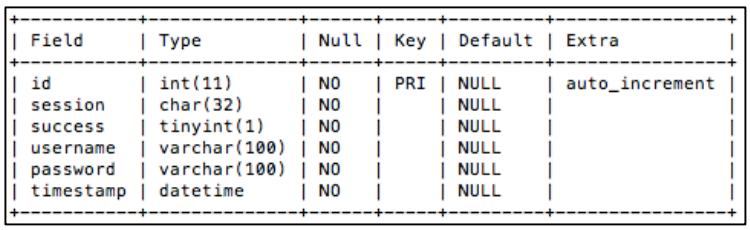
Las tablas de los Honeypots desplegados y que guardan la información a utilizar para crear las reglas son las siguientes:

4.2.1 Tablas Cowrie

En esta sección mostraremos las tablas que almacenan la información relevante relacionada con nuestra propuesta en este tipo de Honeypots.

Las tablas del Honeypot Cowrie son las siguientes:

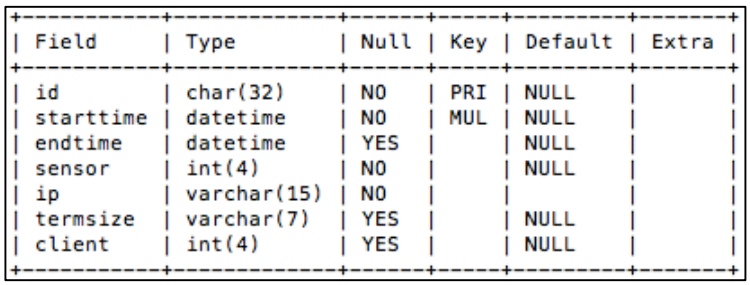
- **auth:** La información relevante para nuestro trabajo almacenada en esta tabla es: el par usuario-contraseña (columnas username y password), el identificador de la sesión asociada al atacante (columna session) y la fecha en la cual se realizó el ataque (columna timestamp). Esta es una de las tablas que utilizaremos para crear nuestras reglas, ya que todas aquellas direcciones que intenten acceder a nuestro sistema por ssh serán estarán ahí almacenadas. En la Ilustración 14 se muestra el esquema de la tabla.



Field	Type	Null	Key	Default	Extra
id	int(11)	NO	PRI	NULL	auto_increment
session	char(32)	NO		NULL	
success	tinyint(1)	NO		NULL	
username	varchar(100)	NO		NULL	
password	varchar(100)	NO		NULL	
timestamp	datetime	NO		NULL	

Ilustración 14. Tabla auth Cowrie.

- **sessions:** La información relevante para nuestro trabajo almacenada en esta tabla es: dirección IP del atacante (columna ip) y el identificador de la sesión asociada al atacante (columna id), ya que nos ayudará a realizar el cruce con la tabla auth. En la Ilustración 15 se muestra el esquema de la tabla.



Field	Type	Null	Key	Default	Extra
id	char(32)	NO	PRI	NULL	
starttime	datetime	NO	MUL	NULL	
endtime	datetime	YES		NULL	
sensor	int(4)	NO		NULL	
ip	varchar(15)	NO			
termsize	varchar(7)	YES		NULL	
client	int(4)	YES		NULL	

Ilustración 15. Tabla sessions Cowrie.

4.2.2 Tablas Glastopf

En esta sección mostraremos las tablas que almacenan la información relevante relacionada con nuestra propuesta en este tipo de Honeypots.

La tabla del Honeypot Glastopf es la siguiente:

- **events:** La información relevante para nuestro trabajo almacenada en esta tabla es: dirección IP de atacante (columna source) y la información que introducen los atacante en la página simulada (columna request_raw). Esta es una de las tablas de las que utilizaremos para crear nuestras reglas, ya que todas aquellas direcciones que

intentan hacer inyección SQL están aquí almacenadas. En la Ilustración 16 se muestra el esquema de la tabla.

Field	Type	Null	Key	Default	Extra
id	int(11)	NO	PRI	NULL	auto_increment
time	varchar(30)	YES		NULL	
source	varchar(30)	YES		NULL	
request_url	varchar(500)	YES		NULL	
request_raw	text	YES		NULL	
pattern	varchar(20)	YES		NULL	
filename	varchar(500)	YES		NULL	
version	varchar(10)	YES		NULL	
sensorid	varchar(36)	YES		NULL	

Ilustración 16. Tabla events Glastopf.

Para más información sobre las tablas de los Honeypots ver Anexo C.

4.3 Infraestructura

En esta sección mostraremos el esqueleto de la infraestructura creada para el procesamiento de la información de la base de datos y la posterior creación e incorporación de reglas de forma automática.

Siguiendo el esquema de la sección de Análisis (Ilustración 12. Esquema proyecto.) mostraremos las diferentes estructuras que se han utilizado para que cada módulo realice su acción.

4.3.1 Uso de los datos de los Honeypots

Los datos recogidos por los Honeypots se encuentran en la base de datos del servidor, en las tablas que hemos mencionado en el apartado anterior. Esta base de datos servirá para recopilar toda la información recogida por los Honeypots.

A principios del mes de junio de 2016 se desplegaron de forma definitiva todos los Honeypots, quedándose todos ellos capturando información de los ataques que reciben. Desde ese momento quedaron "expuestos" en las direcciones IP proporcionadas para ese uso. La fase previa a ésta, la de instalación y configuración de los Honeypots, ha sido realizada totalmente en colaboración con Diego Jurado, mientras que la fase de análisis de los datos almacenados se realizó de forma independiente.

Una de estas direcciones se encuentra fuera del sistema de protección dado por la UAM, mientras que la otra está bajo la seguridad que proporciona la universidad. Disposición ésta ofrecida por uno de los integrantes del sistema de protección para verificar que sus procesos de protección de los sistemas informáticos de la universidad son efectivos.

4.3.2 Procesamiento de la información

Una vez recopilada la información de los Honeypots, se procesará para hallar los diferentes tipos de ataque que han realizan aquellas direcciones registradas en la base de datos del servidor.

Debido a los Honeypots instalados (Cowrie y Glastopf), podremos separar aquellos atacantes por intento de acceso por SSH o por inyección SQL.

Por lo tanto, la información que se almacenará de un atacante en nuestra infraestructura es la siguiente:

- Dirección IP del atacante.
- Tipo de ataque realizado. Valor 'SQL' si ha realizado inyección SQL o 'SSH' si ha intentado acceder por ssh.
- Si el bloqueo ha sido realizado por el usuario (valor 'U') o por la infraestructura (valor 'I').
- El número de ataques que ha realizado el atacante en el sistema. En el caso de que el atacante sea del tipo 'SQL' tendrá el valor 1.

Con toda esta información, tanto de la infraestructura, como del usuario, se podrá comprobar qué atacantes han sido bloqueados, por qué razón y quién ha realizado el bloqueo.

4.3.3 Creación e implantación de reglas

Tras haber conseguido la información que es relevante de los ataques, se continúa con la creación e implantación de las reglas que se generarán. Se ha abordado la creación de reglas que bloquean el acceso de las direcciones IP mediante la utilización de Iptables (ver sección 2.4).

En el sistema se pueden diferenciar varios tipos de reglas dependiendo de la información recogida de la base de datos. A continuación, mostramos los diferentes tipos de reglas que se manejarán en nuestra investigación:

- Reglas que no se incorporarán en el Iptables del sistema:
    - **Tipo TMP:** Se le asignará a aquellas reglas que no hayan superado el umbral máximo de ataques permitidos en el sistema (umbral configurable).
- Reglas que se incorporarán en el Iptables del sistema:
    - **Tipo M1:** Se le asignará a aquellas reglas cuyos atacantes hayan superado el umbral máximo de ataques permitidos en el sistema (umbral configurable) o hayan realizado inyección SQL.
    - **Tipo M2:** Se le asignará a aquellas reglas que han interactuado con ambos tipos de Honeypots de la forma descrita en los requisitos funcionales de la infraestructura.

La información que se manejará de las reglas que se incorporan en el sistema será la siguiente:

- **Regla:** Regla que se incorporará en el sistema.
- **Regla Eliminar:** Regla para eliminar la regla del sistema.
- **Tipo de regla:** Tendrá alguno de los valores TMP, M1 o M2, según las especificaciones mencionadas anteriormente.
- **Estado:** Determina si una regla está activa o no en nuestro sistema. Tomará los valores 'True' o 'False'.
- **Atacante:** Almacenará la información del atacante asociado a la regla creada. Será una estructura como se citó en el apartado anterior (ver sección 4.3.2).
- **Fecha en la que entró la regla en el sistema:** Fecha expresada en segundos en la que la regla entró en nuestro sistema. Este campo nos ayudará a eliminar aquellas

reglas cuyos atacantes no hayan vuelto a interactuar con nuestro sistema durante un período de tiempo determinado.

- **Número de veces en el sistema:** Número de veces que la regla ha estado presente en el sistema. Este campo nos ayudará a la transición del tipo de regla M1 al M2.

Una vez que hemos recogido toda la información de las reglas a aplicar, se incorporarán automáticamente a nuestro sistema todas aquellas que sean del tipo M1 o M2. También cabe la posibilidad de la incorporación de reglas manuales por parte del usuario del sistema.

5. Desarrollo

En esta sección mostraremos cómo se ha desarrollado el sistema para el procesamiento de la información almacenada en la base de datos de los Honeypots, la creación automática de reglas, la incorporación manual de reglas por parte del usuario, la actualización automática de reglas y la visualización de datos relevantes, tanto de reglas, como de atacantes.

Se ha realizado un programa, 'HoneyPRules.py', en el lenguaje de programación Python, que interactúa con el usuario para manejar y manipular las nuevas reglas que se generaros automáticamente.

En primer lugar, se incorporan reglas en el sistema, bien por medio de la importación del último fichero log que se encuentre en el sistema generado por el programa en una ejecución previa, o bien recopilando la información de la base de datos del servidor y generando en ese mismo momento todas las reglas.

Las partes en las cuales se divide el sistema de procesamiento de información y manejo de reglas en nuestro sistema son las siguientes:

5.1 Actualización automática de reglas

En este módulo, el usuario podrá dejar el sistema en modo online, por lo que estará continuamente descargando información de los Honeypots en la base de datos para posteriormente llevar cabo la gestión de las reglas.

El programa permite que las reglas temporales cambien a reglas finales de tipo M1 ó M2, en función de los criterios que se han descrito anteriormente. De esta forma, se incorporarán definitivamente en el sistema y permanecerán en éste hasta la detención del programa junto a las nuevas reglas que se han generado e incorporado de forma automática. Por tanto, todo ello hace que el sistema se proteja mejor de aquellos atacantes que son potencialmente peligrosos, así como estar al día de aquellas direcciones IP de las que debamos protegernos.

En cualquier momento se puede interrumpir el modo online, dejando de actualizar las reglas del sistema utilizando la información de los Honeypots. Esta acción dejará las reglas actualizadas hasta el momento de su detención y exportará la información relevante de éstas, que expondremos más adelante, en un fichero log generado de forma automática por el programa.

En la Ilustración 17 se muestra la salida producida por 'HoneyPRules.py' para cada ciclo de actualización de reglas.

```
Se han actualizado 2 reglas y se han introducido 1 nuevas reglas.
Se han actualizado 0 reglas y se han introducido 2 nuevas reglas.
```

Ilustración 17. Actualización de reglas infraestructura.

5.2 Manejo de las reglas: creación, eliminación y limpieza

En este módulo, el usuario será capaz de realizar acciones de creación, eliminación y limpieza de reglas, tanto de aquellas que han sido añadidas por la infraestructura tras el procesamiento de la información de los ataques, como de las que se han integrado en el sistema de forma manual.

La creación de reglas por parte del usuario puede ser referida a direcciones que se encuentran en nuestro sistema con anterioridad, por estar presente en los datos almacenados en la base

de datos y no sean definitivas (tipo de regla TMP), o direcciones de las que se presuman potenciales ataques.

Adicionalmente, se podrán eliminar todas las reglas que estén en ese momento en el sistema, tanto las creadas manualmente como las generadas de forma automática. Esto permite una mayor libertad de elección acerca de las reglas que el usuario interese que estén en el sistema.

En las Ilustraciones 18 y 19 se muestran los diferentes menús proporcionados por 'HoneyPRules.py' para realizar las acciones sobre las reglas mencionadas anteriormente.

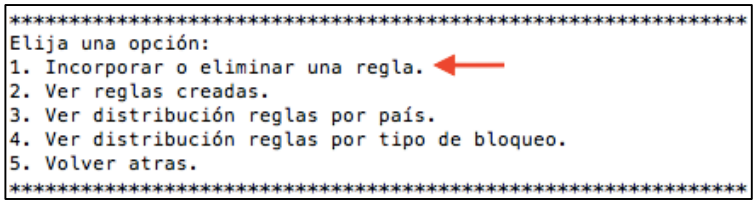


Ilustración 18. Incorporación o eliminación de una regla.

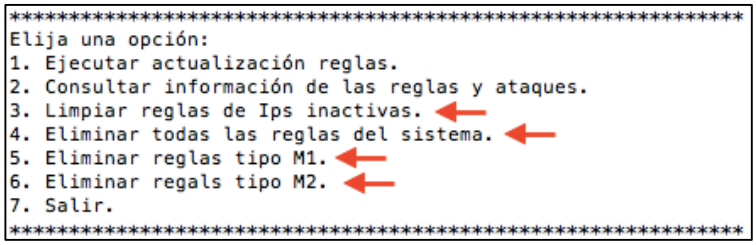


Ilustración 19. Eliminación y limpieza de reglas.

5.3 Visualización de reglas

Este módulo será el encargado de mostrar la información de las reglas que hay en el sistema en cualquier momento, a petición del usuario. Permitirá consultar qué reglas han sido creadas manualmente o por parte de la infraestructura tras el procesamiento de la información almacenada, ambas con un máximo de 20 reglas debido a que, si el volumen de reglas a mostrar es demasiado grande, no sería posible mostrarlas de forma adecuada. Al igual que se pueden visualizar las reglas del sistema, se pueden observar aquellas direcciones que se encuentran disponibles para su bloqueo (tipo TMP).

Adicionalmente, dispondrá de la posibilidad de realizar consultas de las reglas por tipo de ataque o por país de origen de éste.

En la Ilustración 20 se muestra el correcto funcionamiento de la visualización de reglas creadas por la infraestructura de forma automática y generada por 'HoneyPRules.py'. De forma análoga, se realizará la visualización de las reglas creadas por el usuario y la distribución de las reglas por tipo o por situación geográfica.

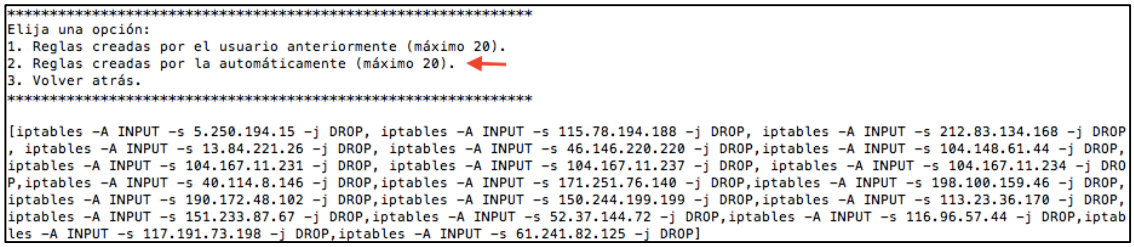


Ilustración 20. Visualización reglas generadas automáticamente.

5.4 Sistema de logs de la infraestructura

Este módulo es el encargado de la entrada y salida de información de las reglas que se han creado en anteriores ejecuciones del programa, como de los procesos realizados durante la ejecución de éste. En los ficheros logs se guardará información relevante sobre las reglas presentes en el sistema en un tiempo determinado, haciendo posible la realización de backups, además de ser útil a la hora de ver la información de las reglas de una forma offline.

La importación de las reglas en el sistema se realizará con el fichero de logs más reciente. En caso de no existir, se realizará una recopilación de los datos almacenados por los Honeypots en el servidor.

La exportación de las reglas se llevará a cabo cuando el usuario salga del modo online, o cuando salga de la ejecución del programa, ya que el usuario puede efectuar acciones sobre las reglas que se hayan añadido por alguno de los dos medios mencionados.

Los ficheros de logs manejarán toda la información almacenada sobre las reglas (ver sección 4.3.3), pero del atacante sólo se guardará el tipo de ataque, el tipo de bloqueo y el número de ataques realizados (ver sección 4.3.2). Esta información es la que tiene mayor relevancia en nuestro sistema a la hora de realizar cualquier función sobre las reglas. Esta información se verá reflejada en el fichero con el orden marcado en la definición de los datos a almacenar de las reglas (Ver sección 4.3.3), a excepción de la información del atacante que irá según el orden mencionado en este párrafo, todo ello separado por el carácter '|'. Un ejemplo de una línea del fichero de logs sería el siguiente: *iptables -A INPUT -s xxx.xxx.xxx.xxx -j DROP|iptables -D INPUT -s xxx.xxx.xxx.xxx -j DROP|TMP|True|1466367621|86400|1|SSH|I|9|*. Por lo que cualquier acción sobre las reglas que se quieran incorporar se puede realizar también a través de los ficheros logs.

En la Ilustración 21 se muestra la extracción de uno de los ficheros de logs generados por 'HoneyPRules.py'.

```
iptables -A INPUT -s 212.129.26.205 -j DROP|iptables -D INPUT -s 212.129.26.205 -j DROP|M1|True|1466367621|86400|1|SSH|I|129|
iptables -A INPUT -s 113.163.6.200 -j DROP|iptables -D INPUT -s 113.163.6.200 -j DROP|M1|True|1466367621|86400|1|SSH|I|27|
iptables -A INPUT -s 212.129.26.206 -j DROP|iptables -D INPUT -s 212.129.26.206 -j DROP|M1|True|1466367621|86400|1|SSH|I|35|
iptables -A INPUT -s 81.39.172.41 -j DROP|iptables -D INPUT -s 81.39.172.41 -j DROP|TMP|True|1466367621|86400|1|SSH|I|9|
iptables -A INPUT -s 5.250.194.15 -j DROP|iptables -D INPUT -s 5.250.194.15 -j DROP|TMP|True|1466367621|86400|1|SSH|I|1|
iptables -A INPUT -s 115.78.194.188 -j DROP|iptables -D INPUT -s 115.78.194.188 -j DROP|TMP|True|1466367621|86400|1|SSH|I|1|
iptables -A INPUT -s 212.83.134.168 -j DROP|iptables -D INPUT -s 212.83.134.168 -j DROP|M1|True|1466367621|86400|1|SSH|I|10|
iptables -A INPUT -s 46.146.220.220 -j DROP|iptables -D INPUT -s 46.146.220.220 -j DROP|TMP|True|1466367621|86400|1|SSH|I|6|
iptables -A INPUT -s 104.148.61.44 -j DROP|iptables -D INPUT -s 104.148.61.44 -j DROP|M1|True|1466367621|86400|1|SSH|I|12|
iptables -A INPUT -s 104.167.11.231 -j DROP|iptables -D INPUT -s 104.167.11.231 -j DROP|M1|True|1466367621|86400|1|SSH|I|15|
iptables -A INPUT -s 104.167.11.237 -j DROP|iptables -D INPUT -s 104.167.11.237 -j DROP|M1|True|1466367621|86400|1|SSH|I|10|
iptables -A INPUT -s 104.167.11.234 -j DROP|iptables -D INPUT -s 104.167.11.234 -j DROP|M1|True|1466367621|86400|1|SSH|I|34|
iptables -A INPUT -s 40.114.8.146 -j DROP|iptables -D INPUT -s 40.114.8.146 -j DROP|TMP|True|1466367621|86400|1|SSH|I|6|
iptables -A INPUT -s 171.251.76.140 -j DROP|iptables -D INPUT -s 171.251.76.140 -j DROP|M1|True|1466367621|86400|1|SSH|I|33|
iptables -A INPUT -s 198.100.159.46 -j DROP|iptables -D INPUT -s 198.100.159.46 -j DROP|TMP|True|1466367621|86400|1|SSH|I|2|
```

Ilustración 21. Extracción fichero de log.

6. Integración, pruebas y resultados

En este apartado mostraremos diferentes pruebas realizadas para mostrar la integración de nuestra infraestructura en un sistema informático, así como los resultados obtenidos tras un periodo de tiempo de ejecución del programa, gracias al que se crean y/o actualizan reglas de bloqueo en el sistema de forma automática y dinámica.

Podemos distinguir para las pruebas de integración los siguientes supuestos de ejecución: ejemplo donde una regla del tipo TMP pase a M1, un ejemplo donde se pase de M1 a M2, y otro donde se puede comprobar el correcto funcionamiento de la incorporación de nuevas reglas al sistema. En todos éstos se seleccionará una dirección que se encuentre almacenada en la base de datos debido a haber realizado ataques a la infraestructura de Honeypots previamente.

6.1 Transición de tipo TMP a M1

Para esta prueba seleccionaremos una dirección de un atacante que haya sido registrada por el sistema y se haya creado la regla correspondiente de tipo TMP previamente. Se producirá el paso al tipo M1 una vez que se haya superado el umbral máximo permitido de número de ataques. El nuevo tipo de regla se modificará en el siguiente ciclo de actualización, parámetro éste configurable por el usuario, pero cuyo valor por defecto es de 30 minutos.

Realizando consultas al servidor en las tablas pertinentes, en este caso la relacionada con el Honeypot Cowrie, seleccionamos una dirección que a fecha 19/06/2016 había interactuado con al menos uno de este tipo de Honeypots de nuestra red sin superar el umbral de ataques, y en una fecha posterior esa misma dirección lo excede. Esto se muestra a continuación en la Ilustración 22.

```
mysql> SELECT b.ip, count(*) as num_attacks from honeynet.auth as a inner join honeynet.sessions as b
    -> on a.session=b.id where b.ip = '103.207.36.177' and a.timestamp<'2016-06-19 00:00:00'  group by b.ip;
+----------------+-------------+
| ip             | num_attacks |
+----------------+-------------+
| 103.207.36.177 |           5 |
+----------------+-------------+
1 row in set (0.03 sec)

mysql> SELECT b.ip, count(*) as num_attacks from honeynet.auth as a inner join honeynet.sessions as b
    -> on a.session=b.id where b.ip = '103.207.36.177' and a.timestamp<'2016-06-24 00:00:00'  group by b.ip;
+----------------+-------------+
| ip             | num_attacks |
+----------------+-------------+
| 103.207.36.177 |          29 |
+----------------+-------------+
1 row in set (0.04 sec)
```

Ilustración 22. Consulta dirección IP prueba 1.

Como podemos observar, el atacante entre el 19 y el 24 de junio de 2016 ha aumentado su número de ataques tanto que supone un potencial peligro para nuestro sistema, por lo que deberá actualizarse su tipo de regla de temporal (TMP) a definitiva, en este caso, M1, quedándose esto reflejado en el Iptables del sistema.

Teniendo los dos estados anteriores, se realiza la carga del fichero log del día 19 de junio de 2016 y, a continuación, se pone nuestro sistema en modo online para que vaya actualizando las reglas de forma automática, en este caso diez segundos por cuestiones puramente experimentales. Este parámetro puede ser configurado por el usuario aunque su valor por defecto sea de 30 minutos. Comprobaremos que la dirección no se encontraba anteriormente en el Iptables y que, tras la actualización generada por la infraestructura después del tiempo marcado sí.

Como se muestra en la Ilustración 23 y en la Ilustración 24, la dirección no se encontraba bloqueada ya que no había realizado un número de ataques suficiente para ello, y que después de que la infraestructura actualiza las reglas pasa a ser bloqueada.

```
root@[redacted]~# iptables -C INPUT -s 103.207.36.177 -j DROP
iptables: Bad rule (does a matching rule exist in that chain?).
root@[redacted]:~# clear
root@[redacted]:~# iptables -C INPUT -s 103.207.36.177 -j DROP
```

```
2018  2016-06-24 08:52:04 : iptables -C INPUT -s 103.207.36.177 -j DROP
2019  2016-06-24 08:52:31 : clear
2020  2016-06-24 08:52:56 : iptables -C INPUT -s 103.207.36.177 -j DROP
2021  2016-06-24 08:53:26 : history
```

Ilustración 23. Comprobación actualización e incorporación de reglas.

```
2087  2016-06-24 08:51:21 : python main.py
2088  2016-06-24 08:53:13 : history
```

Ilustración 24. Verificación del tiempo de ejecución prueba.

6.2 Transición de M1 a M2

Al igual que en la prueba anterior, escogeremos una dirección de un atacante que tenga asociado un regla de tipo M1, es decir, que haya superado el umbral de ataques permitido, y que realice ahora un ataque de inyección SQL en alguno de nuestros Honeypots. Este hecho hará que el tipo de regla cambie su tipo a M2, convirtiéndose. Explicaremos en el apartado de Resultados Obtenidos por qué se ha tenido que incluir de forma manual el ataque de inyección SQL. En la Ilustración 25 y 26 se pueden comprobar que cumplen las condiciones explicadas anteriormente.

```
mysql> SELECT b.ip, count(*) as num_attacks from honeynet.auth as a inner join honeynet.sessions as b
    -> on a.session=b.id where a.timestamp < '2016-06-19 00:00:00' and b.ip='95.141.31.18' group by b.ip;
+--------------+--------------+
| ip           | num_attacks |
+--------------+--------------+
| 95.141.31.18 |          21 |
+--------------+--------------+
1 row in set (0.03 sec)
```

Ilustración 25. Consulta dirección IP Cowrie.

```
mysql> SELECT distinct substring_index(source,':',1)  as ip FROM glaspot.events where (request_raw like '%union%' or request_raw like '%select%'
    -> or request_raw like '%1=1%' or request_raw like '%0=0%') and substring_index(source,':',1) = '95.141.31.18';
+---------------+
| ip            |
+---------------+
| 95.141.31.18  |
+---------------+
1 row in set (0.00 sec)
```

Ilustración 26. Consulta dirección IP Glastopf.

Como vemos, la dirección se encuentra en ambas tablas por lo que se deberá ver reflejado en el fichero de logs (Ilustración 27 y 28) creado por la infraestructura cada vez que hace una actualización de las reglas del sistema.

```
iptables -A INPUT -s 124.224.177.182 -j DROP|iptables -D INPUT -s 124.224.177.182 -j DROP|M1|True|1466367621|86400|1|SSH|I|14|
iptables -A INPUT -s 95.141.31.18 -j DROP|iptables -D INPUT -s 95.141.31.18 -j DROP|M1|True|1466367621|86400|1|SSH|I|21|
iptables -A INPUT -s 31.214.243.39 -j DROP|iptables -D INPUT -s 31.214.243.39 -j DROP|TMP|True|1466367621|86400|1|SSH|I|1|
```

Ilustración 27. Captura log 19/06/2016 prueba 2.

```
iptables -A INPUT -s 124.224.177.182 -j DROP|iptables -D INPUT -s 124.224.177.182 -j DROP|M1|True|1466367621|86400|1|SSH|I|14|
iptables -A INPUT -s 95.141.31.18 -j DROP|iptables -D INPUT -s 95.141.31.18 -j DROP|M2|True|1466771876|86400|1|SSH|I|21|
iptables -A INPUT -s 23.96.243.49 -j DROP|iptables -D INPUT -s 23.96.243.49 -j DROP|M1|True|1466771876|86400|1|SSH|I|31|
```

Ilustración 28. Captura log 24/06/2016 prueba 2.

Observamos que se ha reflejado el cambio de tipo de regla con éxito en los logs de la infraestructura, por lo que se actualizará la regla correspondiente.

6.3 Resultados finales

En este apartado expondremos los resultados obtenidos tras analizar los datos del periodo en el que ha estado activa la infraestructura creando y actualizando reglas en el sistema. Para ello, mostraremos diferentes gráficos que contienen información relacionada con las reglas creadas, dándose la correspondiente justificación de cada comportamiento.

6.3.1 Distribución de reglas según su tipo

A continuación, en la Ilustración 29, mostramos el número de reglas que se han generado durante el período de tiempo en el que toda la infraestructura ha estado activa.

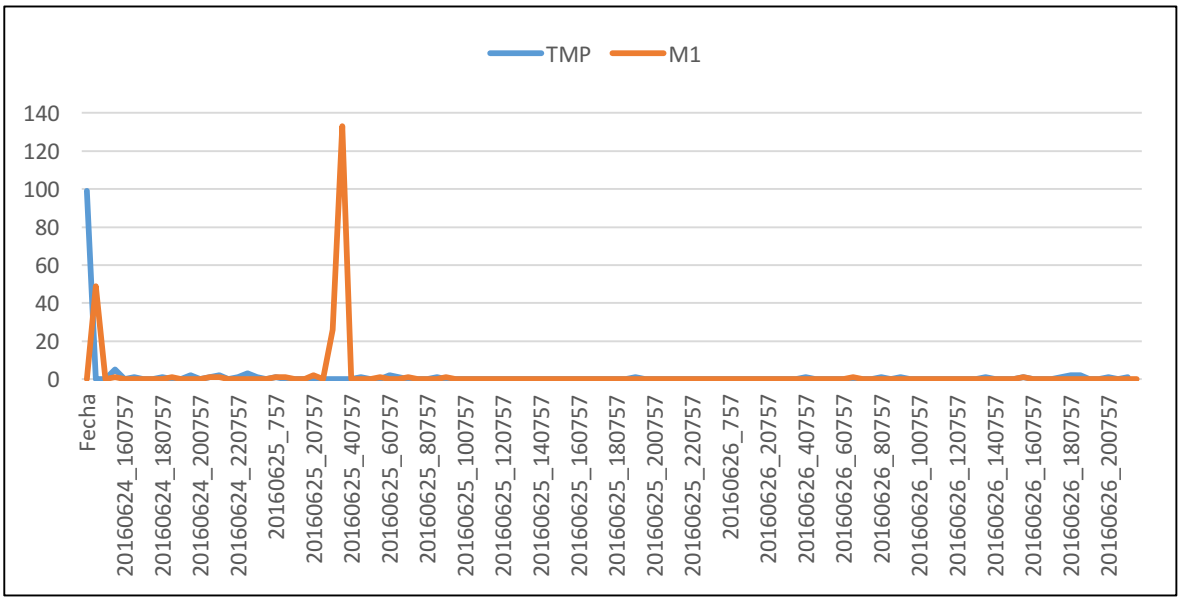


Ilustración 29. Gráfico distribución tipo de reglas.

Como podemos apreciar al comienzo de la incorporación de las reglas, hay un pico para los dos tipos de reglas TMP y M1. Esto se debe a que en el desarrollo del proyecto los Honeypots estuvieron recopilando información mientras que nuestra infraestructura no estaba creando ni incorporando reglas. Esto mismo se vuelve a reflejar el día 25 de junio, ya que se volvió a realizar otra carga de datos en la infraestructura.

También podemos apreciar que la creación de los diferentes tipos de reglas tiene una tendencia monótona que no toma valores muy altos siendo muchas veces cero, esto nos indica que los ataques que se están recibiendo en los diferentes Honeypots son realizados por las mismas direcciones IP que ya había atacado anteriormente.

No se muestran las reglas de tipo M2 porque, tras el periodo el tiempo de recopilación de datos por parte de los Honeypots, no ha habido ningún atacante que haya intentado realizar ataques que se pudiesen clasificar de esa manera. Adicionalmente, se comprobó que la cantidad de direcciones IP que habían interactuado de alguna forma con dos Honeypots diferentes era de cuatro, de lo que podemos intuir que aquellos atacantes que realizan un ataque por ssh no lo hacen vía web por inyección SQL. Aunque no haya sido visto de forma experimental, nuestra infraestructura sí permite detección de este hecho, tal y como se ha comprobado en el ejemplo de transición del tipo de regla de M1 a M2. Por otra parte, si se

aumentase el número de Honeypots en nuestra red, podría darse la aparición de este tipo de regla M2.

6.3.2 Distribución de reglas según el país del atacante

En este apartado mostraremos la distribución de las reglas generadas según la procedencia de los ataques. Esto nos ayudará a prevenir ataques de aquellas direcciones cuyo país de origen sea el que se encuentra con mayor número de reglas asignadas.

A continuación se mostrará en la Ilustración 30 un gráfico que compara la distribución geográfica de las reglas de nuestro sistema al comienzo de la creación de reglas en el sistema frente a la distribución de las mismas una vez eliminadas aquellas cuyas direcciones IP asociadas llevan cierto tiempo (una semana) sin interactuar con alguno de los Honeypots instalados en nuestra red.

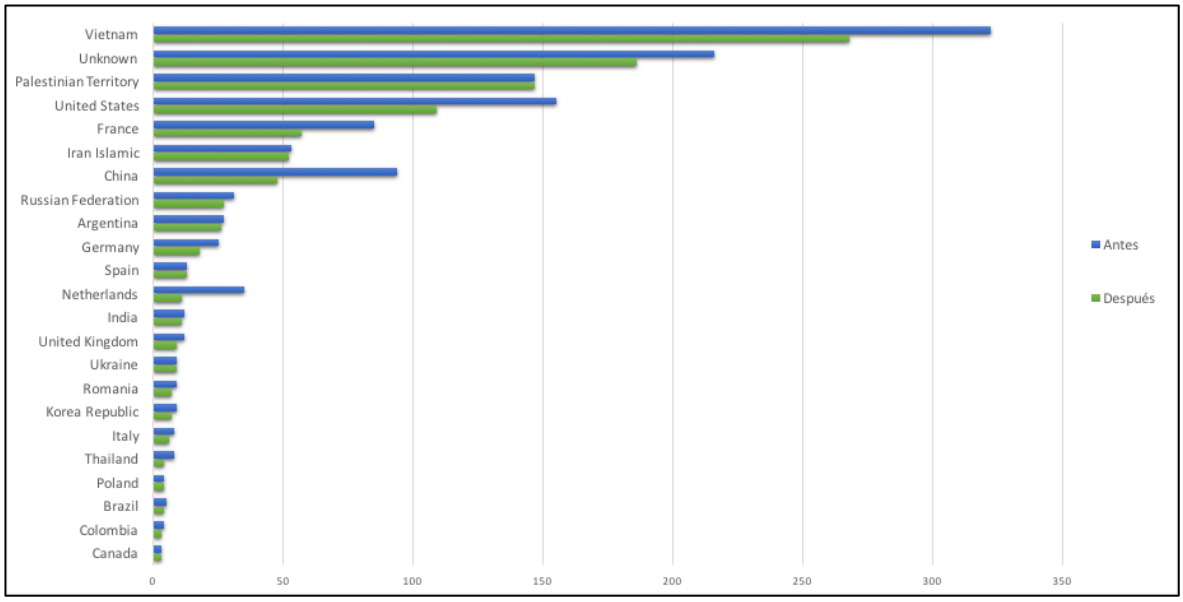


Ilustración 30. Gráfico comparativo distribución geográfica de las reglas.

Como se puede apreciar, se reduce el número de reglas asignas a cada país tras la "limpieza" de reglas. Esto nos ayuda a descongestionar el sistema debido al posible aumento incontrolable de reglas y a eliminar las reglas innecesarias por la inactividad del atacante.

También podemos comprobar que los países que más realizan ataques durante el tiempo a nuestra red son Vietnam, Palestina, Estados Unidos, Francia, Irán y China, porque continúan en las posiciones más altas aunque descartemos reglas por inactividad. Con ello tendremos otro criterio más que permita bloquear mejor direcciones procedentes de esos países.

6.3.3 Limitaciones del estudio

Dado que el sistema donde se incorporan las reglas se encontraba bajo la protección del propio sistema de seguridad de la UAM, siendo éste muy efectivo frente a los ataques registrados por los Honeypots Cowrie y Glastopf, tal y como queda reflejado en el TFG de Diego Jurado Pallares. Como consecuencia, el análisis realizado para comprobar el aumento de la seguridad después de la incorporación de reglas no nos dará evidencias del aumento de la seguridad del sistema.

Por tanto, planteamos la siguiente situación para comprobar la efectividad de nuestra infraestructura: tomar todos los ataques sufridos por los Honeypots desde la fecha en la cual

se comienza a incorporar reglas en el sistema, 23 de junio de 2016, hasta el 27 de junio de 2016, como si fuesen los atacantes del sistema en cuestión, distinguiendo qué tipo de reglas habrían generado tras su paso por nuestra infraestructura, observando así cuántos del total habrían sido bloqueados. De esta forma podemos comprobar la efectividad de las reglas incorporadas en el sistema.

Como muestra la Ilustración 31, se registraron un número total de 2801 ataques, de los cuales 1847 habrían generado reglas del tipo M1, haciendo que la dirección del atacante quede bloqueada; y el resto, 964, habrían generado reglas del tipo TMP, por lo que éstos no sufrirían bloqueo alguno. Estos datos mencionados nos indican que nuestra infraestructura habría bloqueado el 66% de los ataques recibidos, siendo éste un nivel aceptable para el aumento de la seguridad del sistema. Pudiendo ser este porcentaje mayor si relajásemos los umbrales para la generación de reglas por los cuales se bloquean las direcciones IP.

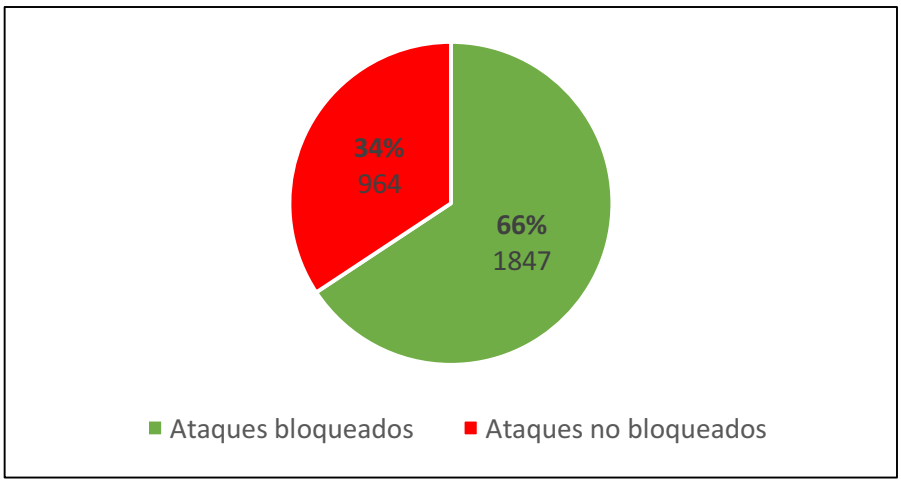


Ilustración 31. Gráfico eficiencia infraestructura.

7. Conclusiones, discusión y trabajo futuro

7.1 Conclusiones y discusión

En este proyecto se ha llevado a cabo la creación de una infraestructura que recoge información sobre los ataques sufridos por los Honeypots para su posterior procesamiento, creando reglas para Iptables de forma automática y dinámica que se incorporan en el sistema donde se encuentre instalada la infraestructura. Esto prevendría ataques futuros, lo cual tiene como consecuencia el aumento de la seguridad en el sistema. Por tanto, hemos cumplido el objetivo principal fijado al comienzo de nuestro proyecto.

A continuación, mostraremos las conclusiones referentes a los otros objetivos que nos marcamos al principio del proyecto:

- **Conclusión Objetivo 1:** Este objetivo estaba relacionado con el estudio de las herramientas Honeypots actuales en una distribución de máquina virtual (VM VirtualBox). Gracias al haber cumplido este objetivo, habiendo realizado el estudio de estas herramientas, y habiendo comprendido su funcionamiento, se decidió qué tipo de Honeypots y sistemas operativos se debían utilizar para nuestro proyecto.

- **Conclusión Objetivo 2:** En referencia a este objetivo, se realizó con éxito el despliegue de los diferentes Honeypots seleccionados en la red de la UAM tras la finalización de la fase de análisis marcada en el objetivo 1.

- **Conclusión Objetivo 3:** En lo referente al procesamiento de la información y creación de reglas, se ha conseguido procesar la información almacenada por los Honeypots y la generación de reglas de forma automática y dinámica conforme la información que van recopilando los Honeypots instalados va actualizándose. Para el desarrollo de este objetivo nos hemos ayudado de la implementación de un programa, 'HoneyPRules.py', en el lenguaje Python debido a sus ventajas en cuanto al manejo de información y las librerías que tiene ya implementadas.

Por consiguiente, podemos decir que hemos cumplido los objetivos que nos marcamos al inicio de la realización del proyecto.

Tras la exposición de los resultados obtenidos, mostrados en la sección anterior, podemos deducir que las reglas creadas e incorporadas por la infraestructura son efectivas. Por tanto, hacen que la seguridad del sistema donde se instale aumente. Esto se debe a que los Honeypots al estar continuamente expuestos a ataques, hacen que nuestra infraestructura detecte direcciones altamente peligrosas más rápidamente, pudiendo así proteger el sistema antes de ser atacado. Adicionalmente, analizando los resultados obtenidos, podemos mencionar que unos de los tipos de regla (tipo M2) tiene una caracterización muy estricta, ocasionando que, hasta el momento, no haya ninguna regla de ese tipo que hay sido creada por la infraestructura.

En cuanto a la interactuación entre la infraestructura y un usuario cualquiera, se implementó un programa que permite, de una forma sencilla e intuitiva, que el usuario pueda crear una configuración personalizada de las reglas de su sistema, siendo esta funcionalidad muy útil para la prueba de funcionamiento de la infraestructura.

Para concluir, debemos hacer una discusión entre el proyecto realizado y aquel que tomamos como referencia. Las principales ventajas de nuestra propuesta, y que son novedosas respecto de la otra investigación, son: la creación de reglas de forma automática y dinámica; la utilización de varios tipos de Honeypots, sin límite en el número de éstos, para recopilar información a utilizar en la creación de reglas; la posibilidad de realizar una configuración de las reglas presentes en el sistema; y la posibilidad de eliminar aquellas reglas que lleven cierto tiempo sin interactuar con los diferentes Honeypots instalados.

7.2 Trabajo futuro

A continuación indicaremos las posibles líneas de investigación que puede tomar el proyecto tras el estudio de los resultados obtenidos y de las diferentes herramientas utilizadas en este proyecto.

Los trabajos futuros en los que puede derivar este proyecto son los siguientes:

- Estudio de otras tipologías de ataque que almacenan los Honeypots instalados en este proyecto, más concretamente Glastopf, para la incorporación y actualización de reglas.

- Realización de diferentes reglas con la información almacenada de los Honeypots ya instalados en este proyecto, además del bloqueo de direcciones IP, incorporar el redireccionamiento de IP y el bloqueo de IP por un puerto concreto.

- Realizar el estudio en un sistema que se encuentre fuera de la influencia de filtros de direcciones para poder ver la actuación de la infraestructura sobre éste.

- Integración de la infraestructura creada con un IDS, como SNORT [16], para ver su integración con otros otros sistemas los resultados que generan.

- Añadir información adicional a manejar por nuestra infraestructura mediante la instalación Honeypots diferentes a los presentes en ésta.

Referencias

[1] Loras R. Even, “Honey Pot Systems Explained”, 2000
[2] William W. Martin, “Hone y Pots and Honey Nets - Security through Deception”, SANS Institute, pp 1-8, 2001
[3] The Honeynet Project, http://www.honeynet.org/
[4] Vinod Yegneswaran, Paul Barford and Vern Paxson, “Using Honeynets for Internet Situational Awareness”, The ICSI Networking and Security Group, pp 1-6
[5] Honeypot, Intrusion Detection, Incident Response, http://www.honeypots.net/
[6] Wikipedia, Honeypot (computing), Sección Types
[7] Honeydrive, https://bruteforce.gr/honeydrive
[8] Michel Oosterhof, http://www.micheloosterhof.com/cowrie/
[9] Lukas Rist (auth), Sven Vetsch, Marcel Koßin and Michael Mauer (co-auth), “A dynamic, low-interaction web application honeypot”, November 2014, pp 1-2.
[10] Netfilter/iptables Project, http://netfilter.org/
[11] Wikipedia, Iptables, https://en.wikipedia.org/wiki/Iptables, https://es.wikipedia.org/wiki/Netfilter/iptables
[12] Ashley Thomas, “Adaptive Real Time Intrusion Detection Systems”, pp 11-32, 2002
[13] Dave Shackleford, “Real-Time Adaptive Security”, SANS Institute, pp 1-10, 2008
[14] Anazida Zainal, Mohd Aizaini Maarof, Siti Mariyam Shamsuddin and Ajith Abraham, “Design of Adaptive IDS with Regulated Retraining Approach”, pp 590-599
[15] Radu State, “Adaptative and Selfconfigurable Honeypots”, University of Luxembourg, pp 1- 25, 9th September 2011
[16] James Kipp, “Using Snort as an IDS and Network Monitor in Linux”, GIAC, pp 1-4, 2001
[17] Gérard Wagener. “Self-Adaptive Honeypots Coercing and Assessing Attacker Behaviour.”, Institut National Polytechnique de Lorraine - INPL, 2011.
[18] Wikipedia, Game Theory, https://en.wikipedia.org/wiki/Game_theory
[19] Wikipedia, Teoria de Juegos, https://es.wikipedia.org/wiki/Teoría_de_juegos
[20] Richard Hammer (auth) and Joey Niem (adv), “Enhancing IDS using, tiny honeypot”, SANS Institute, pp3-27, 2006.
[21] Diego Jurado Pallares, “Análisis y estudio de Honeypots complejos: Honeynets”, Trabajo Fin de Grado, 2016.

Glosario

Honeypot: Es todo aquel sistema que sirve de señuelo diseñado para atraer a un atacante potencial y quitar su atención de los sistemas críticos

IDS: Sistema de detección de intrusos.

IP: Protocolo de Internet

Reglas de bloqueo: Son reglas que se introducirán dentro de nuestro sistema y que llevarán a cabo el bloqueo de la comunicaciones que se reciba por parte del atacante que figura en la regla. Son tipos de reglas las temporales (TMP) y las permanentes (M1 y M2).

Iptables: Componente de Netfilter que se ofrece al usuario mediante la línea de comandos se permite configurar las tablas encargadas del filtrado de los paquetes que recibe el sistema informático así como las cadenas y reglas que almacena cada una de la tablas

Cowrie: Tipo de Honeypot que registra los ataques recibidos por SSH.

Glastopf: Tipo de Honeypot que registra los ataques recibidos a través de la simulación de una página web vulnerable a ser atacada.

Anexos

A Manual Honeypot Kippo (red doméstica)

Tras haber estado buscando información relacionada con la instalación y configuración de este tipo de Honeypot observando que la información para ello está en diferentes lugares. Se decidió resumir y crear un manual para la instalación y configuración del Honeypot Kippo, aunque se haya utilizado Cowrie en este TFG las funciones básicas de ambos son similares, en una red doméstica situándolo en un DMZ (zona desmilitarizada), ya que encontrar la información relevante a veces es complicado.

El primer paso a realizar es la instalación del programa Oracle VM VirtualBox pudiéndonoslo descargar su última versión para cualquier sistema operativo. En caso de que estemos en Ubuntu podremos descargarlo e instalarlo desde el Centro de Software de Ubuntu.

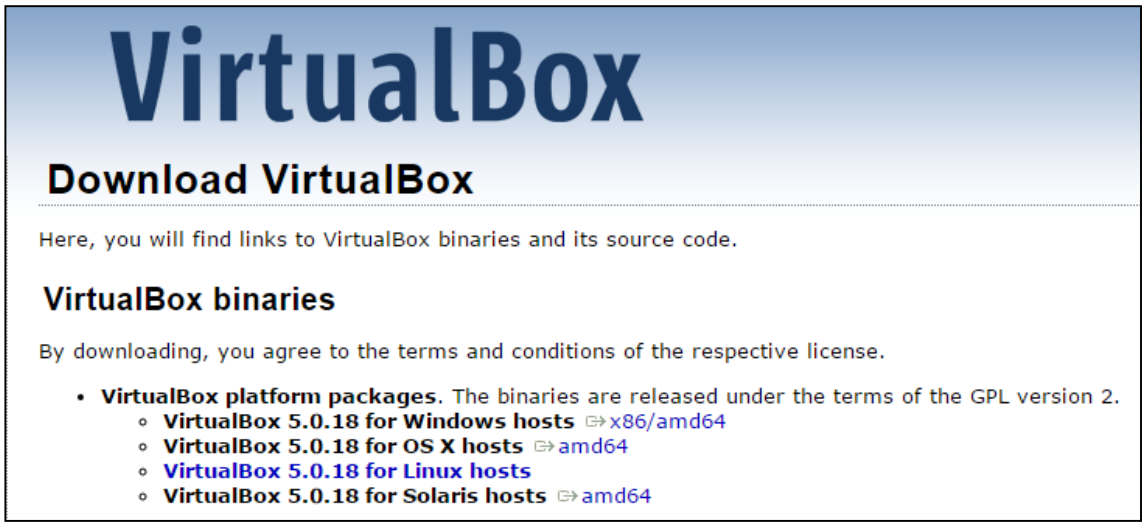


Ilustración 32. Sección Descargas Oracle VM VirtualBox

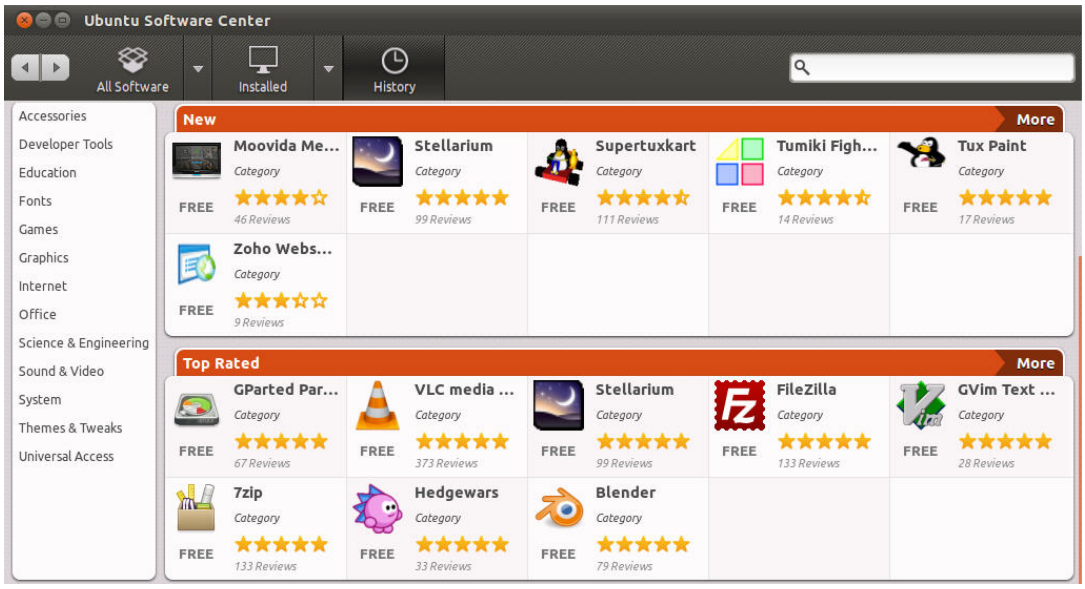


Ilustración 33. Centro de software de Ubuntu

A continuación, una vez instalado VM VirtualBox, nos descargamos la imagen de Honeydrive, en la cual están las Honeypots que vamos a configurar además de muchas más que se pueden instalar en la máquina virtual para que reciban ataques dependiendo del tipo de Honeypot que sea.

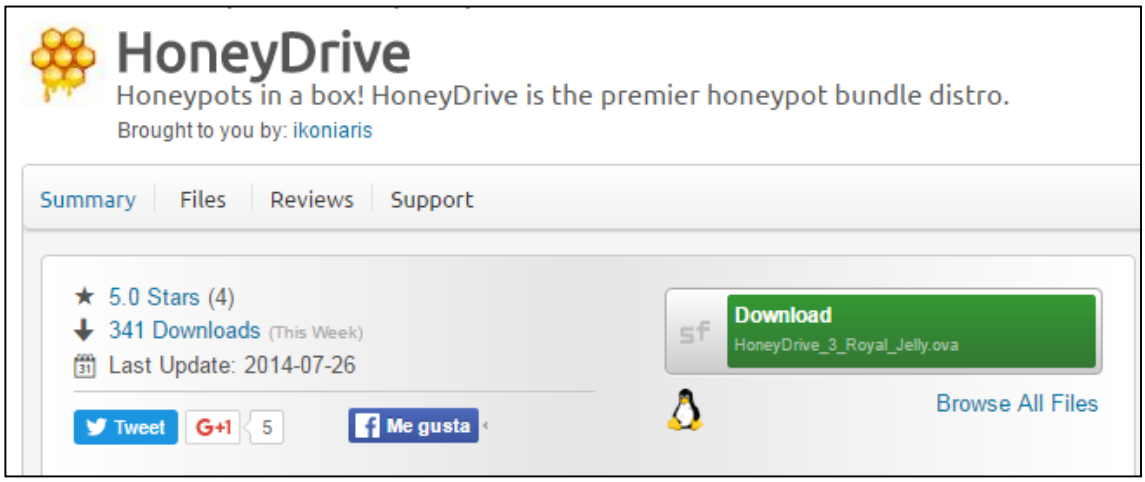


Ilustración 34. Descarga Honeydrive

Después hacemos doble click en el fichero que se nos ha descargado (HoneyDrive 3 Royal Jelly edition) y se enlazará directamente con el VirutalBox. Nos saltará la siguiente pantalla.

Ilustración 35. Configuración Honeydrive

Podremos las opciones iguales a la imagen mostrada.

El proceso de carga de Honeydrive con VirtualBox tarda bastante tiempo, entre 30-40 minutos aproximadamente. Una vez se haya completado la importación nos introducimos en

la configuración de la máquina virtual y modificamos las opciones de la sección *Red* y poner la placa de red de modo promiscuo.

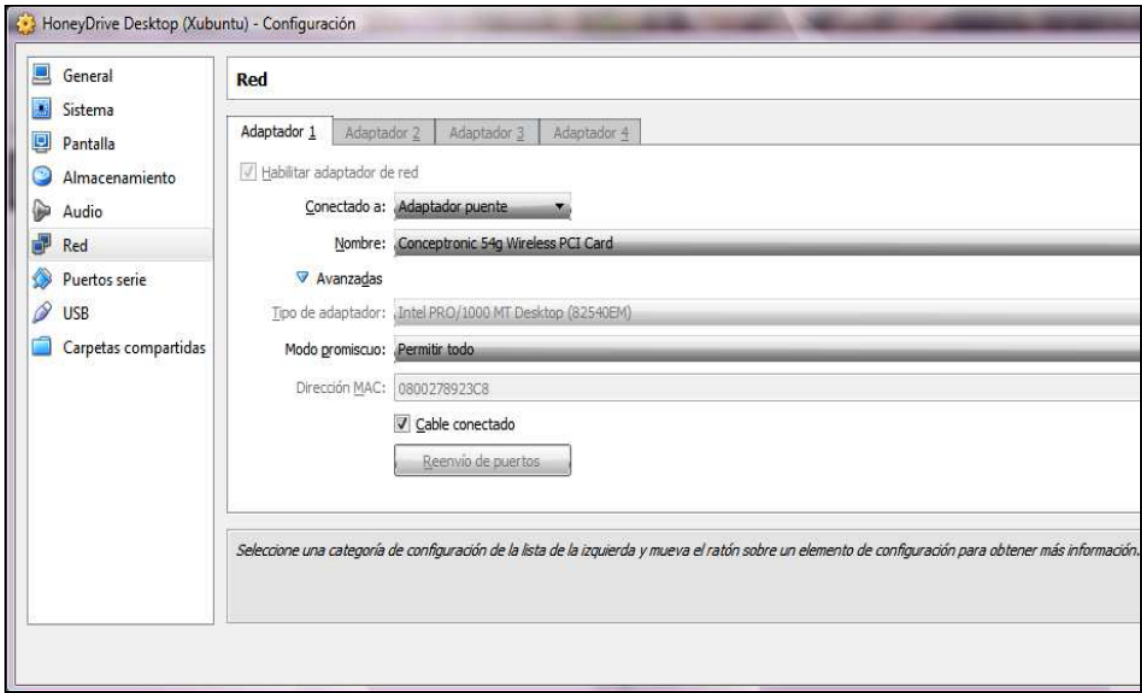


Ilustración 36. Configuración Sección Red Honeydrive

Iniciamos la máquina virtual y nos aparecerá la siguiente pantalla:

Ilustración 37. Login Honeydrive

Introducimos la contraseña: honeydrive. Esta la podremos modificar mediante el comando sudo passwd en la terminal más adelante.

Finalmente nos aparecerá el Escritorio de la máquina virtual:

Ilustración 38. Escritorio Honeydrive

En el fichero readme.txt aparecen todas las Honeypots que se pueden instalar en la máquina virtual, localizando todos los elementos de cada una de ellas dentro de la máquina virtual así como las contraseñas que se necesitan en ellas (contraseñas por defecto) si son necesarias para acceder a algo de la Honeypot instalada.

```
[System]
Connectivity:          DHCP
Hostname:              honeydrive
User:                  HoneyDrive
Username/password:     honeydrive/honeydrive
Sudo password:         honeydrive
Log in automatically:  enabled

[Virtualization]
VBox Guest additions:  installed
Shared Clipboard:      bidirectional
Drag'n'Drop:           disabled

[LAMP]
Apache 2 support:      PHP, Perl, Python, Ruby/Rails
Document root:         /var/www/
Apache 2 changes:      AllowOverride All (/var/www/), ServerTokens Minimal, ServerSignature Off
Apache php.ini changes:        max_execution_time = 300
                       max_input_time = 180
                       memory_limit = 256M
                       post_max_size = 256M
                       upload_max_filesize = 256M
                       max_file_uploads = 40
MySQL root password:   honeydrive

[Kippo]
Location:              /honeydrive/kippo/
Start script:          /honeydrive/kippo/start.sh
Stop script:           /honeydrive/kippo/stop.sh
Downloads:             /honeydrive/kippo/dl/
TTY logs:              /honeydrive/kippo/log/tty/
Credentials:           /honeydrive/kippo/data/userdb.txt
MySQL database:        kippo
MySQL user/password:   root/honeydrive

[Kippo-Graph]
Location:              /var/www/kippo-graph/
Configuration:         /var/www/kippo-graph/config.php
URL:                   http://local-or-remote-address/kippo-graph/
MySQL database:        kippo
MySQL user/password:   root/honeydrive

[Kippo-Malware]
Location:              /honeydrive/kippo-malware/
```

Ilustración 39. Honeydrive readme.txt

A partir de aquí ya entramos más explícitamente en la configuración y presentación del Honeyppot Kippo.

Este tipo de Honeypot nos va a permitir emular un servicio SSH con un usuario y contraseña poco seguras (ya que queremos recibir ataque y ver que hacen los atacantes cuando acceden a nuestro sistema), reproducir un sistema de ficheros en el que incluso se puede integrar un sistema operativo y ejecutar comandos.

En este tipo de Honeypot podremos ver exactamente que ha hecho el atacante, debido a que se registran todos aquellos comandos que ha ido introduciendo en la terminal, cuando ha podido acceder a nuestro sistema. Además tiene una aplicación web que nos muestra mediante diferentes graficas datos de los ataques recibidos, junto con una sección que nos muestra la localización geográfica del ataque.

Mostramos los directorios importantes en este Honeypot:

- dl/: carpeta donde se guardan los ficheros descargados mediante wget.
- log/kippo.log: fichero donde se guarda información de uso y depuración.
- log/tty/: carpeta donde se guardan los log de las sesiones de los diferentes atacantes.
- utils/playlog.py: herramienta para reproducir en la terminal los logs de sesión.
- utils/createfs.py: se utilizará para crear el fs.pickle, que ayudará a crear ficheros, carpeta y demás en la Honeypot para camuflarla y que el atacante no se dé cuenta de que no se trata de una Honeypot.
- fs.pickle: falso sistema de ficheros.
- honeyfs/: contenido del falso sistema de ficheros. Podremos crear una copia de un sistema real.

Localización de los diferentes ficheros de Kippo:

[Kippo] Script: /honeydrive/kippo/start.sh Downloads: / honeydrive /kippo/dl/ TTY logs: / honeydrive /kippo/log/tty/ Credentials: / honeydrive /kippo/data/userdb.txt MySQL database: kippo MySQL user/password: root/honeydrive	[Kippo-Graph] Location: /var/www/kippo-graph/ Config: /var/www/kippo-graph/config.php URL: http://local-or-remote-IP-address/kippo-graph/ MySQL database: kippo MySQL user/password: root/honeydrive
[Kippo2MySQL] Location: / honeydrive /kippo2mysql/ MySQL database: kippo2mysql MySQL user/password: root/honeydrive	[Kippo-Scripts] Location: / honeydrive /kippo-scripts/ + kippo-sessions + kippo-stats + kippo2wordlist

Aquí nos muestra la disposición de carpetas y ficheros del Honeypot Kippo.

A continuación debemos abrir una terminal, que está situada en el escritorio de la máquina virtual llamada 'Terminator'.

Pero antes de poner en marcha el Honeypot, vamos a indicar cómo modificar la contraseña que bien por defecto en el servicio SSH. Escribimos en la terminal el siguiente comando:

cd / honeydrive /kippo/data

Dentro de este directorio podemos observar que hay un fichero llamado userdb.txt, este es el fichero que debemos modificar si queremos cambiar el usuario y/o la contraseña del servicio SSH del Honeypot. Para ello introduciremos el siguiente comando:

nano userdb.txt

Podremos ver que los valores por defecto para el usuario y contraseña son 'root' y '123456', así modificaremos, o no, los valores. Además podremos introducir nuevos valores en líneas sucesivas a la primera que hay en el fichero, añadiendo así nuevos usuarios que se puedan acceder a nuestra Honeypot.

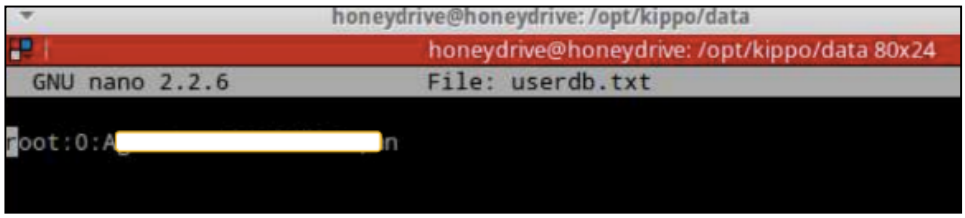


Ilustración 40. Configuración usuarios Kippo.

Los cambios los guardaremos presionando las teclas CTR+X y a continuación Y para guardarlo definitivamente.

Ahora viene la parte más importante. Hay varias formas para montar una Honeypot en una DMZ en una red doméstica, aquí explicaremos la que se ha utilizado en las pruebas del TFG.

A.1 Creación DMZ

Lo que debemos hacer en introducir en la terminal el siguiente comando:

ifconfig

Nos quedamos con la información que viene en la sección de inet, dirección a la cual no estamos conectado a internet, que comenzará con 192.168.1.XXX (XXX será un número entre 1 y 255). Esta será la dirección IP en la cual se creará la DMZ.

Ahora accederemos a nuestro router introduciendo en nuestro navegador 192.168.1.1. Introducimos el usuario y la contraseña, por lo que accederemos a nuestro router para poder crear la DMZ. Algunas compañías de teléfono tienen la posibilidad de crear una DMZ y abrir puertos mediante una página web propia, debe informarse si puede hacerlo. Dentro de las opciones del router hay una sección llamada DMZ, accedemos a esa sección y creamos una DMZ introduciendo la dirección IP 192.168.1.XXX. Después accederemos a la sección de abrir los puertos ahí abriremos el puerto 22, utilizado por el Honeypot Kippo para recibir los ataques, en la misma dirección que hemos estado mencionando en todo momento.

Por último ya sólo queda ponerla en marcha para poder ver los resultados más adelante. Para ello ejecutaremos en la terminal los siguientes comandos:

cd /honeydrive/kippo/

chmod +x ./start.sh /* este comando sólo debemos ejecutarlo en caso de que no nos deje hacer ./start.sh */

./start.sh

Ya tenemos creada y en funcionamiento nuestra Honeypot Kippo.

Para verificar que todo está en funcionamiento podemos introducir el siguiente comando en la terminal:

sudo netstat -atnp | grep 22

Si todo esta correcto nos saldrá lo siguiente:

```
honeydrive@honeydrive:/opt/kippo$ sudo netstat -atnp | grep 2
2
[sudo] password for honeydrive:
tcp        0      0 0.0.0.0:22              0.0.0.0:*
       LISTEN      2139/python
```

Ilustración 41. Escucha de ataques Kippo.

Ya tenemos todo puesto en marcha por lo que sólo hace falta esperar a recibir ataques Podemos consultar la base de datos para mirar los ataques, comandos introducidos, etc. O podemos visualizarlo en el entorno gráfico de Kippo, Kippo-Graph.

Para poder ponerlo en marcha, simplemente introducir en el navegador la url que nos indica el fichero de configuración "readme.txt":

http://local-or-remote-IP-address/kippo-graph/ /* en la parte local-or-remote-IP-address pondremos localhost */

Si abrimos KIPPO-GRAPH veremos si hemos sufrido ataques, cuando se han producido, IPs del ataque, gráficos generados por este entorno con los datos que almacena Kippo, comandos que ha introducido ele atacante cuando ha accedido al Honeypot, etc.

Ilustración 42. Pantalla inicial Kippo-Graph.

B Manual Honeypot Glastopf (red doméstica)

Tras haber estado buscando información relacionada con la instalación y configuración de este tipo de Honeypot observando que la información para ello está en diferentes lugares. Decidí resumir y crear un manual para la instalación y configuración del Honeypot Glastopf, utilizada en este TFG, en una red doméstica situándolo en un DMZ (zona desmilitarizada), ya que encontrar la información relevante a veces es complicado.

El primer paso a realizar es la instalación del programa Oracle VM VirtualBox pudiéndonoslo descargar su última versión para cualquier sistema operativo. En caso de que estemos en Ubuntu podremos descargarlo e instalarlo desde el Centro de Software de Ubuntu.

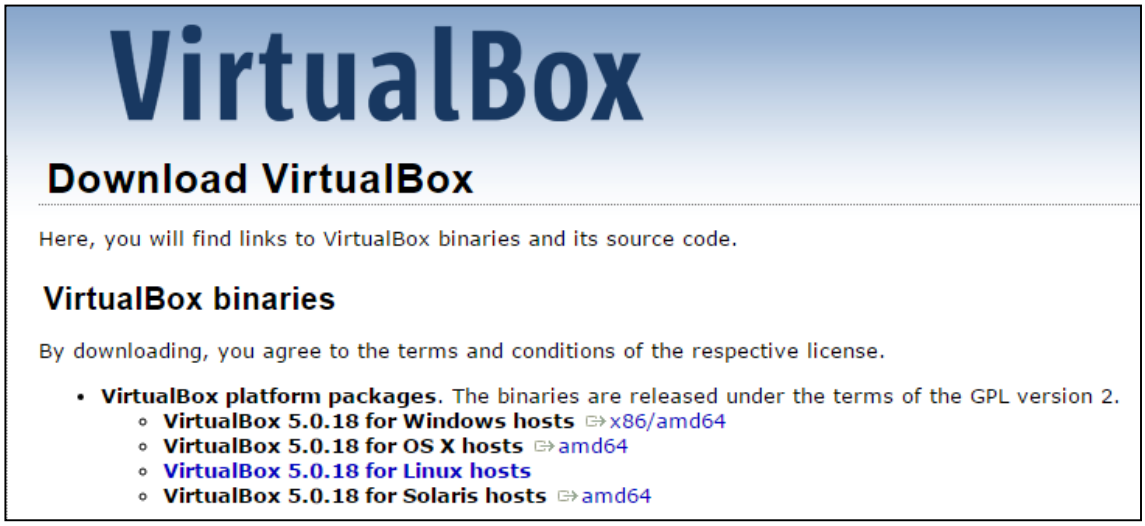


Ilustración 43. Sección Descargas Oracle VM VirtualBox

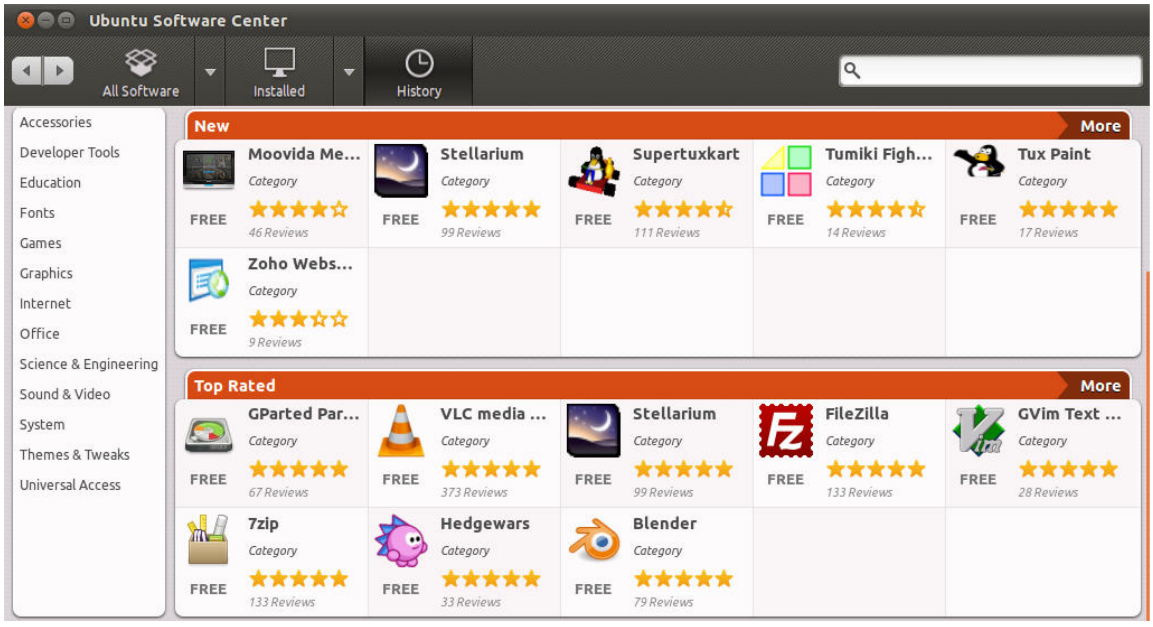


Ilustración 44. Centro de software de Ubuntu.

A continuación podemos instalarla de dos formas:

- Instalando previamente Honeydrive (explicado en la Sección Manual Honeypot Kippo (red doméstica)), ya que están todos los ficheros, scripts y base de datos que se necesitan para la ejecución de esta Honeypot.

  Seguir los pasos indicados en A.1 Creación DMZ, en este caso deberemos abrir el puerto 80 de nuestro router porque es el cual utiliza este tipo de Honeypot para recibir ataques desde el exterior.

  Después de toda la configuración de la DMZ deberemos abrir una terminal, llamada Terminator en Honeydrive, y ejecutar los siguiente comandos para poner en funcionamiento la Honeypot Glastopf:

  cd /honeydrive/glastopf/bin/

  chmod +x glastopf-runner

  sudo glastopf-runner

  A partir de aquí pasamos a: B.1 Comprobación de la instalación.

- Instalando una máquina virtual con el sistema operativo Xubuntu 14.04 (mismo sistema operativo que tiene Honeydrive) para tener un sistema íntegro para el Honeypot.

  Seguir los pasos indicados en Creación DMZ, en este caso deberemos abrir el puerto 80 de nuestro router porque es el cual utiliza este tipo de Honeypot para recibir ataques desde el exterior.

  A continuación introduciremos los siguientes comandos en una terminal para que se instalen y creen todos aquellos componentes que necesitamos para funcione nuestra Honeypot.

  /* Instalación de todos los componentes que necesita Glastopf */
  sudo apt-get update
  sudo apt-get install python2.7 python-openssl python-gevent libevent-dev python2.7- dev build-essential make
  sudo apt-get install python-chardet python-requests python-sqlalchemy python-lxml
  sudo apt-get install python-beautifulsoup mongodb python-pip python-dev python-setuptools
  sudo apt-get install g++ git php5 php5-dev liblapack-dev gfortran libmysqlclient-dev
  sudo apt-get install libxml2-dev libxslt-dev
  sudo pip install --upgrade distribute

  /* Instalación BFR */
  cd /opt
  sudo git clone git://github.com/glastopf/BFR.git
  cd BFR
  sudo phpize
  sudo ./configure --enable-bfr
  sudo make && sudo make install
  echo "zend_extension = /usr/lib/php5/20090626+lfs/bfr.so" >> /etc/php5/cli/php.ini

```
/* Instalación de Glastopf */
sudo pip install glastopf
cd /opt
sudo mkdir myhoneypot
cd myhoneypot
sudo glastopf-runner

/* Es posible que nos de un fallo el anterior comando debido a la versión utilizada de
greenlet */
sudo pip install --upgrade greenlet
```

Si aún así continua el error, abrimos el fichero glastopf.cfg.dist y comentamos las siguientes líneas para que no se ejecute esa parte del código porque puede que no esté operativo actualmente.

```
[hpfeed]
enabled = True          /* esta línea la modificamos con valor False, ya que
                           sino sigue dando error */
host = hpfriends.honeycloud.net
port = 20000
secret = 3wis3l2u5l7r3cew
# channels comma separated
chan_events = glastopf.events
chan_files = glastopf.files
ident = x8yer@hp1
```

A partir de aquí pasamos a: B.1 Comprobación de la instalación.

B.1 Comprobación de la instalación

Esta parte es común en ambas configuraciones, debido a que ayuda a comprobar que todo funciona correctamente y sólo se diferencia en las rutas en donde lo comprobaremos.

Este script no acaba la ejecución, es decir, no para de ejecutar ya que está a la espera de atacantes. Una vez haya atacado alguien aparecerá en esta terminal.

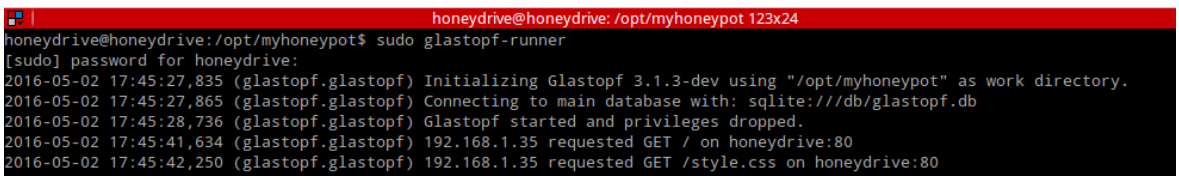


Ilustración 45. Ejecución script Glastopf.

Para ver que está a la espera de ataques abrimos una nueva terminal e introducimos el siguiente comando:

sudo netstat -nat | grep 80

Apareciendo lo siguiente en la terminal:

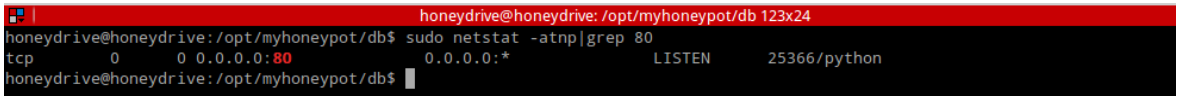


Ilustración 46. Puerto 80 abierto Glastopf.

Como podemos comprobar está escuchando lo que suceda en el puerto 80.

Para comprobar que está operativo podemos introducir desde cualquier navegador de internet la dirección IP de la máquina virtual y nos aparecerá la siguiente página:

Ilustración 47. Página Honeypot Glastopf.

Esta página ira cambiando su título cada vez se refresque o cada vez que accedamos a ella. Al introducir datos y pulsar el botón *Submit* aparecerá esa interacción en la terminal que se ha quedado ejecutando el script glastopf-runner, además de introducirse en la base de datos propia del Honeypot en la tabla *events*.

C Diagramas bases de datos Honeypot

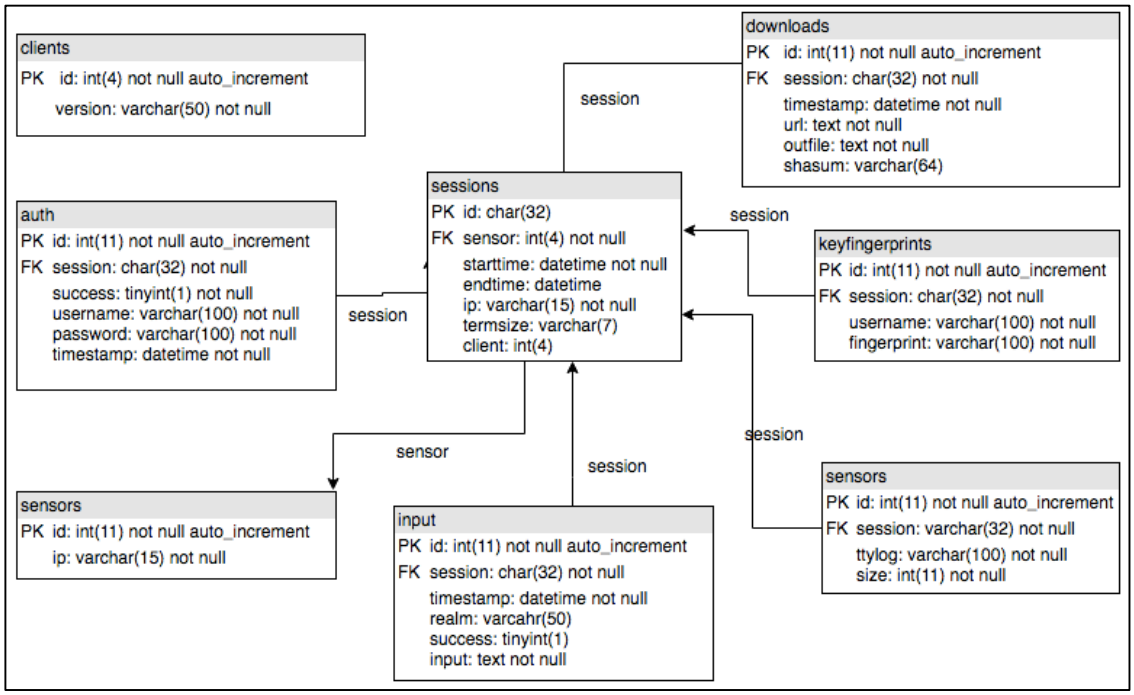


Ilustración 48. Diagrama base de datos Honeypot Cowrie.

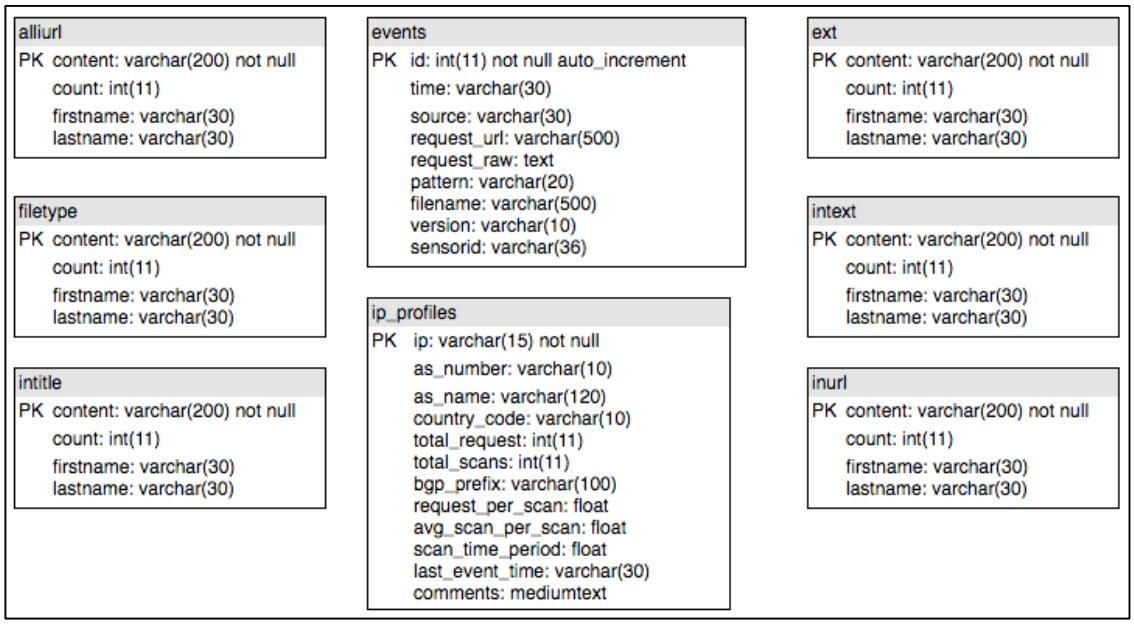


Ilustración 49. Diagrama base de datos Glastopf.

D Código ejecución de la infraestructura

A continuación incorporamos los códigos utilizados para la creación automática y dinámica de reglas.

1. Código principal (HoneyPRules.py).

```
# -*- coding: utf-8 -*-
from attacksLib import *
from sqlDataBase import *
from menus import *

''' Importacion de la librerias para la ejecucion de comandos '''
import os # Libreria par ejecutar comandos en local
import time # Libreria para poder acceder al reloj
import commands # Libreria para ejecutar comandos y recoger la salida
import pxssh # Libreria para poder conectarnos por ssh

logsPath=os.path.dirname(os.path.abspath(__file__))+'/logs/'
print logsPath
from os import listdir
from os.path import isfile, join
onlyfiles = [f for f in listdir(logsPath) if isfile(join(logsPath, f))]

''' Creamos un diccionario que almacenará los atacantes '''
attackersDict={}
reglas={}

''' Recogemos todas las IPs que han atacado nuestro sistema '''
allIpAttackers=getAllIpAttack()

''' Recopilamos todas la reglas generadas automáticamente '''
if onlyfiles:
  getAttacksToBlock(logsPath+sorted(onlyfiles)[-1], attackersDict, reglas)
else:
  getAttacksToBlock('', attackersDict, reglas)

while True:
        printMainMenu()
        selection=raw_input()
        if selection == '5':
                print
                print "Saliendo del sistema."
                break
        selectionMenu(selection, attackersDict, reglas)
while True:
        print "¿Quiere guardar todos los cambios realizados? (s/n):)"
        print
        sel=raw_input()
        if sel == 's':
                exportLog(reglas)
                break
```

```
        elif sel == 'n':
                break
        else:
                print "Opción no válida."
                print
print
print "Hasta pronto."
          deleteAllIptables(reglas)
```

2. Código de interactuación entre las reglas y ataques (attacksLib.py)

```
# -*- coding: utf-8 -*-
import time # Libreria para poder acceder al reloj
import os # Libreria par ejecutar comandos en local
from sqlDataBase import *
import commands

# Variables de diferenciación de los diferentes ataques
UMBRAL_M1=10 # Numero de ataques que debe realizar un ataque para que se
materialice la regla de forma permanente
UMBRAL_M2=2 # Numero de sistemas maximo que puede atacar
TIME_ACTIVE=24*60*60  # Tiempo que esta activa una regla, 24h
TIME_PURGE = 7*24*60*60 # Si un atacante no interactua con nuestro sistema en una
semana, la regla asociada a este se elimina

''' Definimos la estructura para almacenar la información de los atacantes '''
class Attacker(object):
    def __init__(self,ipAttacker, typeAttack, blockBy, numAttacks):
        self._ipAttacker = ipAttacker
        self._typeAttack = typeAttack
        self._blockBy = blockBy
         self._numAttacks = numAttacks

    @property
    def ipAttacker(self):
        return self._ipAttacker

    @property
    def typeAttack(self):
        return self._typeAttack

    @property
    def blockBy(self):
        return self._blockBy

    @property
    def numAttacks(self):
        return self._numAttacks

''' Definimos la estructurade las reglas
```

```
'''
class Rule(object):
    def __init__(self, rule, ruleTemp, typeRule, active, attacker, timestamp, ttl, numSys):
        self._rule = rule
         self._ruleTemp = ruleTemp
        self._typeRule = typeRule
        self._active = active
         self._attacker = attacker
         self._timestamp = timestamp
         self._ttl = ttl
         self._numSys = numSys

    @property
    def rule(self):
        return self._rule

    @property
    def ruleTemp(self):
        return self._ruleTemp

    @property
    def typeRule(self):
        return self._typeRule

    @property
    def active(self):
         return self._active

    @property
    def attacker(self):
        return self._attacker

    @property
    def timestamp(self):
        return self._timestamp

    @property
    def ttl(self):
        return self._ttl

    @property
    def numSys(self):
         return self._numSys


def getAttacksToBlock(logPath, attackersDict={}, reglas={}):
    # Si hay algún fichero de log anteriormente cargamos las reglas que hay en el
    if logPath:
         importLog(logPath, reglas, attackersDict)
    # En caso contrario reneramos las reglas a partir de los datos de la base de datos
```

```
    # o en el caso de que el log este vacio
    if not reglas:
        dateEnterSys = time.strftime("%s")
        SSHAttacks = attackSSH(time.strftime("%Y-%m-%d %H:%M:%S"))
        SQLAttacks = attackInjection()
        attacksBothSys = getAttackBothSys()
        numM1 = 0
        numM2 = 0
        numTemp = 0
        ''' Creacion de las reglas de bloqueo para las IP de SSH
        '''
        for ipAttackerSSH in SSHAttacks:
        regla='iptables -A INPUT -s '+ ipAttackerSSH[0] +' -j DROP'
        p = Attacker(ipAttackerSSH[0], 'SSH', 'I', ipAttackerSSH[1])
        attackersDict[p.ipAttacker]=p
                numSys = 1
                if ipAttackerSSH[1] >= UMBRAL_M1:
                        typeRule = 'M1'
                        numM1 = numM1 + 1
                elif ipAttackerSSH[0] in attacksBothSys:
                        typeRule = 'M2'
                        numSys  = UMBRAL_M2
                        numM2 = numM2 + 1
                else:
                        typeRule = 'TMP'
                        numTemp = numTemp + 1
                ruleTemp = 'iptables -D INPUT -s '+ ipAttackerSSH[0] +' -j DROP'
                r = Rule(regla, ruleTemp, typeRule, 'True', p, dateEnterSys, TIME_ACTIVE,
numSys)
                insertUpdateDict(p.ipAttacker, attackersDict, reglas, r, p, False)

        for ipAttackerSQL in SQLAttacks:
                numSys = 1
                insertUpdate = False
                if ipAttackerSQL[0] in attacksBothSys:
                    typeRule = 'M2'
                    numSys  = UMBRAL_M2
                        insertUpdate = True
                        numM2 = numM2 + 1
                regla='iptables -A INPUT -s '+ ipAttackerSQL[0] +' -j DROP'
        p = Attacker(ipAttackerSQL[0], 'SQL', 'I', 1)
        attackersDict[p.ipAttacker]=p
                ruleTemp = 'iptables -D INPUT -s '+ ipAttackerSQL[0] +' -j DROP'
        r = Rule(regla, ruleTemp, 'M1', 'True', p, dateEnterSys, TIME_ACTIVE, numSys)
                insertUpdateDict(p.ipAttacker, attackersDict, reglas, r, p, insertUpdate)

        exportNumRulesCreate(numTemp, numM1, numM2)
        exportLog(reglas)

''' Funcion que actualia las reglas
```

```
'''
def updateInfoRules(attackersDict, reglas):
    dateEnterSys = time.strftime("%s")
    num_insert = 0
    num_update = 0
    numM1 = 0
    numM2 = 0
    numTemp = 0
    SSHAttacks = attackSSH(time.strftime("%Y-%m-%d %H:%M:%S"))
    SQLAttacks = attackInjection()
    attacksBothSys = getAttackBothSys()
    ''' Creacion de las reglas de bloqueo para las IP de SSH
    '''
    for ipAttackerSSH in SSHAttacks:
        ip = ipAttackerSSH[0]
        insertUpdate = ip in reglas.keys()
        if insertUpdate:
                regla = reglas[ip]
                ruleTemp = regla.ruleTemp
            numSys = regla.numSys
            typeRule = regla.typeRule
            timestamp = regla.timestamp
              if not regla.attacker.blockBy == 'U':
                    blockBy = 'I'
                    if regla.typeRule == 'TMP':
                            numAttacks = ipAttackerSSH[1]
                            num_attacks_update = (numAttacks -
long(regla.attacker.numAttacks))
                            if numAttacks >= UMBRAL_M1 and num_attacks_update > 0:
                                    typeRule = 'M1'
                                    timestamp = dateEnterSys
                                    num_update = num_update + 1
                                    numM1 = numM1 + 1
                    elif regla.typeRule == 'M1':
                            if ip in attacksBothSys:
                                    typeRule = 'M2'
                              timestamp = dateEnterSys
                              num_update = num_update + 1
                                    numSys = UMBRAL_M2
                                    numM2 = numM2 + 1
                else:
                        blockBy = regla.blockBy
        else:
                num_insert = num_insert + 1
                if ipAttackerSSH[1] >= UMBRAL_M1:
                        typeRule = 'M1'
                        numM1 = numM1 + 1
                elif ip in attacksBothSys:
                        typeRule = 'M2'
                        numM2 = numM2 + 1
```

```
                else:
                        typeRule = 'TMP'
                        numTemp = numTemp + 1
                ruleTemp = 'iptables -D INPUT -s '+ ipAttackerSSH[0] +' -j DROP'
                timestamp = dateEnterSys
                blockBy = 'I'
                numSys = 1
        p = Attacker(ip, 'SSH', blockBy, ipAttackerSSH[1])
        r = Rule('iptables -A INPUT -s '+ ip +' -j DROP', ruleTemp, typeRule, 'True', p,
timestamp, TIME_ACTIVE, numSys)
        insertUpdateDict(ip, attackersDict, reglas, r, p, insertUpdate)

    for ipAttackerSQL in SQLAttacks:
        ip = ipAttackerSQL[0]
        insertUpdate = ip in reglas.keys()
        if not insertUpdate:
                num_insert = num_insert + 1
                ruleTemp = 'iptables -D INPUT -s '+ ip +' -j DROP'
                numSys = 1
                typeRule = 'M1'
                timestamp = dateEnterSys
                numM1 = numM1 + 1
        else:
                typeRule = reglas[ip].typeRule
                ruleTemp = reglas[ip].ruleTemp
                timestamp = reglas[ip].timestamp
                numSys = reglas[ip].numSys
                if ip in attacksBothSys:
                        num_update = num_update + 1
                        typeRule = 'M2'
                        timestamp = dateEnterSys
                        numSys = UMBRAL_M2
                        numM2 = numM2 + 1
        p = Attacker(ip, 'SQL', 'I', 1)
       r = Rule('iptables -A INPUT -s '+ ip +' -j DROP', ruleTemp, typeRule, 'True', p,
timestamp, TIME_ACTIVE, numSys)
        insertUpdateDict(ip, attackersDict, reglas, r, p, insertUpdate)

    exportNumRulesCreate(numTemp, numM1, numM2)
    print "Se han actualizado " + str(num_update) + " reglas y se han introducido " +
str(num_insert) + " nuevas reglas."
    exportLog(reglas)

''' Funcion para actualizar o inserta en nuestro diccionario de reglas
'''
def insertUpdateDict(ip, attackersDict, reglas, regla, attacker, insertUpdate):
        if insertUpdate:
                if not reglas[ip].typeRule == regla.typeRule and not regla.typeRule == 'TMP':
                        deleteOneRule(regla.ruleTemp)
                        addRule(regla.rule)
```

```
                del reglas[ip]
                del attackersDict[ip]
        reglas[ip] = regla
        attackersDict[ip] = attacker

''' Funcion que importa reglas del fichero de log
'''
def exportLog(reglas):
    f =
open(os.path.dirname(os.path.abspath(__file__))+'/logs/log'+time.strftime("%Y_%m_%d_
%H%M%S")+'.txt', 'w')
    for regla in reglas.keys():
        numSys = str(reglas[regla].numSys)
        ttl = str(reglas[regla].ttl)
        numAttacks = str(reglas[regla].attacker.numAttacks)
        print>>f,
reglas[regla].rule+"|"+reglas[regla].ruleTemp+"|"+reglas[regla].typeRule+"|"+reglas[regla]
.active+"|"+reglas[regla].timestamp+"|"+ttl+"|" \

+numSys+"|"+reglas[regla].attacker.typeAttack+"|"+reglas[regla].attacker.blockBy+"|"+nu
mAttacks+"|"
    f.close()
    print "Se han exportado "+str(len(reglas))+" al fichero log."

''' Funcion para importar reglas del fichero log
'''
def importLog(logPath, reglas={}, attackersDict={}):
    f = open(logPath, 'r')
    document=f.readlines()
    linesDocument = ''.join(str(ineAll) for ineAll in document)
    for lineAll in document:
        lineAlmost = lineAll.split('|')
        ip=lineAlmost[0].split(' ')[4]
        p = Attacker(ip, lineAlmost[7], lineAlmost[8], lineAlmost[9])
        attackersDict[p.ipAttacker]=p
        r = Rule(lineAlmost[0], lineAlmost[1], lineAlmost[2], lineAlmost[3], p,
lineAlmost[4], lineAlmost[5], lineAlmost[6])
        reglas[p.ipAttacker]=r
    f.close()
    print "Se han importado "+str(len(reglas))+" reglas del último fichero log con nombre "+
logPath.split('/')[-1]
    addIptables(reglas)

''' Funcion para eliminar aqeullas reglas que han cesado su ataque en nuestro sistema
'''
def cleanRules(attackersDict, reglas):
    numDeleteRules = 0
    lastAttacksQuery = getLastAttack(reglas.keys())
    for ip in lastAttacksQuery:
        regla = reglas[ip[0]]
```

```
        lastAttack=ip[1]
        timeBetweenDates = abs(lastAttack - long(regla.timestamp))
        if timeBetweenDates >= TIME_PURGE and not regla.attacker.blockBy == 'U' and
not regla.typeRule == 'TMP' and not regla.active == 'False':
                r = Rule(regla.rule, regla.ruleTemp, regla.typeRule , 'False', regla.attacker,
regla.timestamp, regla.ttl, regla.numSys)
                insertUpdateDict(regla.attacker.ipAttacker, attackersDict, reglas, r,
regla.attacker, True)
                deleteOneRule(regla.ruleTemp)
                numDeleteRules = numDeleteRules + 1
    print "Se han eliminado " + str(numDeleteRules) + " reglas debido a la inactividad de los
atacantes."

''' Funcion que bloquea una ip dada por el usuario
'''
def blockIp(attackersDict, reglas, ip):
    p=Attacker(ip, ' ', 'U', 1)
    insertUpdate = ip in reglas.keys()
    if insertUpdate:
        regla = reglas[ip]
        r = Rule(regla.rule, regla.ruleTemp, ' ' , 'True', p, regla.timestamp, regla.ttl,
regla.numSys)
    else:
        r = Rule('iptables -A INPUT -s '+ ip +' -j DROP', 'iptables -D INPUT -s '+ ip +' -j
DROP', ' ', 'True', p, time.strftime("%s"), TIME_ACTIVE, 1)
    insertUpdateDict(ip, attackersDict, reglas, r, p, insertUpdate)

''' Funcion que elimina una regla del sistema
'''
def deleteRule(ip, reglas):
        regla = reglas[ip]
        r = Rule(regla.rule, regla.ruleTemp, regla.typeRule, 'False', regla.attacker,
regla.timestamp, regla.ttl, regla.numSys)
        del reglas[ip]
        reglas[ip] = r
        deleteOneRule(regla.ruleTemp)

''' Funcion que elimina una regla del sistema
'''
def deleteOneRule(regla):
        ip = regla.split(' ')
        if 'Bad' not in commands.getstatusoutput('iptables -C INPUT -s ' + ip[4] + ' -j
DROP')[1]:
                os.system(regla)
        else:
                os.system("clear")

''' Funcion que elimina unas reglas del iptables
'''
def deleteIptables(reglas):
```

```
        rulesToDelete = [reglas[ip].ruleTemp for ip in reglas if reglas[ip].active == 'False']
        for rule in rulesToDelete:
                deleteOneRule(rule)

''' Funcion que elimina todas la reglas de iptables que hemos introducido nosotros
'''
def deleteAllIptables(reglas):
        rulesToDelete = [reglas[ip].ruleTemp for ip in reglas if reglas[ip].active == 'True'
and not reglas[ip].typeRule == 'TMP']
       for rule in rulesToDelete:
             deleteOneRule(rule)
        print "Borradas todas las reglas."

''' Funcion que introduce un regla en el sistema
'''
def addRule(regla):
        ip = regla.split(' ')
        if 'Bad' in commands.getstatusoutput('iptables -C INPUT -s ' + ip[4] + ' -j DROP')[1]:
                os.system(regla)
        else:
                print "La regla se encontraba ya en el sistema." + ip[4]

''' Funcion que introduce unas regla en el iptables
'''
def addIptables(reglas):
        rulesToAdd = [reglas[ip].rule for ip in reglas if reglas[ip].active == 'True' and not
reglas[ip].typeRule == 'TMP']
        for rule in rulesToAdd:
                addRule(rule)


''' Funcion que devuelve la distribución de numero de reglas por pais
'''
def getDistCountry(ips):
        return getDistCountrySQL(ips)

''' Funcion que exporta el numero de reglas creado en cada actualizacion
'''
def exportNumRulesCreate(numTemp, numM1, numM2):
        date = time.strftime("%Y%m%d;%H%M%S")
        f = open(os.path.dirname(os.path.abspath(__file__))+'/updatedAttacks.txt', 'a')
       print>>f, str(numTemp) + ";" + str(numM1) + ";" + str(numM2) + ";" + date
                f.close()

''' Funcion que elimina las reglas tipo M1
'''
def deleteM1(reglas):
            rulesToDelete = [reglas[ip].ruleTemp for ip in reglas if reglas[ip].typeRule ==
          'M1']
            for rule in rulesToDelete:
```

```
            deleteOneRule(rule)
        print "Se han eliminado las reglas de tipo M1"


''' Funcion que elimina las reglas tipo M2
'''
def deleteM2(reglas):
        rulesToDelete = [reglas[ip].ruleTemp for ip in reglas if reglas[ip].typeRule ==
      'M2']
        for rule in rulesToDelete:
            deleteOneRule(rule)
```

3. Código interactuación con la base de datos (sqlDatBase.py)

```
# -*- coding: utf-8 -*-

import MySQLdb # Libreria para poder realizar consultas en la bate de datos
import time


''' Variables de conexion de la base de datos '''
DB_HOST='XXX.XXX.XXX.XXX'
DB_USER='yyyyyy'
DB_PASS='zzzzzz'
DB_NAME='aaaaa'

''' Consultas para sacar las direcciones a bloquear '''

''' Consulta para sacar aquellos que han atacado mas de 10 veces y para aquellos que hayan
accedido al sistema
'''
sqlSSH_v2_1 = "SELECT b.ip, count(*) as num_attacks from honeynet.auth as a inner
join honeynet.sessions " \
      +"as b on a.session=b.id where a.timestamp <= '"
sqlSSH_v2_2 = "' group by b.ip;"


''' Consultas para sacar aquellas direcciones que han hecho inyeccion sql en la pagina de
glastopf
'''
sqlInjection = "SELECT distinct substring_index(source,':',1)  as ip FROM glaspot.events
where request_raw like '%union%' or request_raw like '%select%' "\
          +"or request_raw like '%1=1%' or request_raw like '%0=0%'"+";"

''' Incorporamos todas las consultas en comun para no repetir direcciones
'''
sqlUnion="SELECT distinct e.ip from ( ("+ sqlInjection.replace(';',' ') +") union ("+
sqlSSH_fin.replace(';',' ') + ")) as e;"

''' Consulta para sacar todas las Ips de lso atacantes
```

```
'''
sqlIpCowrie="(SELECT DISTINCT b.ip from honeynet.auth as a inner join
honeynet.sessions as b on a.session=b.id)"
sqlIpGlastopf="(SELECT DISTINCT substring_index(source,':',1)  as ip from
glaspot.events)"
sqlAllIp="SELECT DISTINCT c.ip FROM ( "+ sqlIpCowrie +" union "+ sqlIpGlastopf +"
) as c;"


''' Consulta que devuelve la fecha en segundos del ultimo ataque que ha realizado una
atacante
'''
sqlLastAttack_1 = "SELECT b.ip, max(unix_timestamp(a.timestamp)) from honeynet.auth
as a inner join honeynet.sessions as b on a.session=b.id where b.ip in ( '"
sqlLastAttack_2 = " ) group by b.ip;"

''' Consulta que devuelve la distribucion reglas por pais
'''
sqlCountry_1 = "SELECT country, count(*) as num_attackers from honeynet.geo where ip
in ( '"
sqlCountry_2 = " ) group by country;"
sqlCountry_3 = "SELECT country from honeynet.geo where ip in ( '"
sqlCountry_4 = " );"

''' Consulta par ver que iphan atacado a los dossistemas
'''
sqlBothSys = "SELECT * FROM (SELECT DISTINCT b.ip from honeynet.auth as a inner
join honeynet.sessions as b on a.session=b.id) as a inner join (SELECT DISTINCT
substring_index(source,':',1)  as ip from glaspot.events) as b on a.ip=b.ip;"

''' Funcion para ejecutar las consultas a la base de datos '''
def run_query(query=''):
    datos =[DB_HOST,DB_USER,DB_PASS,DB_NAME]
    conn=MySQLdb.connect(*datos)
    cursor=conn.cursor()
    cursor.execute(query)
    ''' Diferenciamos entre las consultas que devuelven datos de las que solamente se
ejecutan'''
    if  query.upper().startswith('SELECT'):
        data=cursor.fetchall()
    else:
        conn.commit()
        data=None
    ''' Cerramos las conexiones '''
    cursor.close()
    conn.close()
    ''' devolvemos el valor tras realizar al consulta '''
    return data

''' Funcion que devuelve los ataques que siguen las condiciones definidas para su bloqueo
```

```
'''
def attacks():
    return run_query(sqlUnion)

''' Funcion que devuelve todas la direcciones Ip almacenadas en el sistema
'''
def getAllIpAttack():
    return run_query(sqlAllIp)

''' Funcion que devuelve los ataques de SSH
'''
def attackSSH(date):
    sqlSSH_final = sqlSSH_v2_1 + date + sqlSSH_v2_2
    return run_query(sqlSSH_final)

''' Funcion que devuelve los ataques de SQL Injection
'''
def attackInjection():
    return run_query(sqlInjection)

''' Funcion que devuelve la fecha del ultimo ataque realizado por un atacante
'''
def getLastAttack(ips):
        sqlLastAttack_fin = sqlLastAttack_1
        for ip in ips:
                sqlLastAttack_fin = sqlLastAttack_fin + ip + "', '"
        return run_query(sqlLastAttack_fin[:-3] + sqlLastAttack_2)

''' Funcion que devuelve el pais y el numero de reglas en el sistema
'''
def getDistCountrySQL(ips):
        sqlCountry_fin = sqlCountry_1
        for ip in ips:
                sqlCountry_fin = sqlCountry_fin + ip + "', '"
        return run_query(sqlCountry_fin[:-3] + sqlCountry_2)

''' Funcion que devuelve los atacantes que ha atacado a ambos sistemas
'''
def getAttackBothSys():
        return [ip[0] for ip in run_query(sqlBothSys)]

‘’’ Funcion que devuelve la distribución de las reglas por país
‘’’
def getDistCountrySQL_2(ips):
       sqlCountry_fin = sqlCountry_3
       for ip in ips:
              sqlCountry_fin = sqlCountry_fin + ip + "', '"
                return run_query(sqlCountry_fin[:-3] + sqlCountry_4)
```

4. Código de interactuación con la infraestructura (menus.py)

```
# -*- coding: utf-8 -*-
from sqlDataBase import *
import attacksLib
import time
import threading
import os
‘’’ Funcion que actualiza las reglas de forma periódica
‘’’
def updateAttacks(attackersDict, reglas):
    global thread
    thread = threading.Timer(1800.0, updateAttacks, [attackersDict, reglas])
    thread.start()
    attacksLib.updateInfoRules(attackersDict, reglas)
‘’’ Funcion que para la actualización automática de las reglas
‘’’
def stopThread(reglas):
    thread.cancel()
    print "Parando actualizacion reglas."
    attacksLib.exportLog(reglas)

''' Funciones de impresion del menu principal
'''
def printMainMenu():
        print
        print "*****************************************************************"
        print "Elija una opción:"
        print "1. Ejecutar actualización reglas."
        print "2. Consultar información de las reglas y ataques."
        print "3. Limpiar reglas de Ips inactivas."
        print "4. Eliminar todas las reglas del sistema."
        print "5. Eliminar reglas tipo M1."
        print "6. Eliminar reglas tipo M2."
        print "7. Salir."
        print "*****************************************************************"
        print

‘’’ Funcion de improesion del primer menú
‘’’
def printPrimaryMenu():
        print
        print "*****************************************************************"
        print "Elija una opción:"
        print "1. Incorporar o eliminar una regla."
        print "2. Ver reglas creadas."
        print "3. Ver distribución reglas por país."
        print "4. Ver distribución reglas por tipo de bloqueo."
        print "5. Volver atras."
        print "*****************************************************************"
        print
```

```
''' Funciones de impresion del segundo menu
'''
def printSecondMenu():
        print
        print "*****************************************************************"
        print "Elija una opción:"
        print "1. Reglas creadas por el usuario anteriormente (máximo 20)."
        print "2. Reglas creadas por la automáticamente (máximo 20)."
        print "3. Volver atrás."
        print "*****************************************************************"
        print

''' Funciones de impresion del primer menu
'''
def printFirstMenu():
        print
        print "*****************************************************************"
        print "Elija una opción:"
        print "1. Ver Ips sin bloquear en el sistema (máximo 20)."
        print "2. Ver Ips bloqueadas por el sistema y manuales. (máximos 20)"
        print "3. Introducir IP a bloquear."
        print "4. Eliminar una  regla."
        print "5. Volver atrás."
        print "*****************************************************************"
        print

''' Funcion de impresion del menu para bloquear una IP manualmente
'''
def printMenuBlockIp():
        print
        print "*****************************************************************"
        print " Introduca la IP a bloquear (XXX.XXX.XXX.XXX) y pulse Enter."
        print " las XXX son números comprendidos del 1 al 255"
        print "*****************************************************************"
        print

''' Funcion de impresion del menu para eliminar una regla manualmente
'''
def printMenuDeleteRule():
        print
        print "*****************************************************************"
        print " Introduca la IP (XXX.XXX.XXX.XXX) de la regla a eliminar."
        print " las XXX son números comprendidos del 1 al 255"
        print "*****************************************************************"
        print


''' Funcion que comprueba la IP introducida por el usuario
'''
def checkIp(ip, reglas):
```

```
  if ip in reglas.keys():
      print "Se procederá a bloquear la IP introducida que se encontraba en el sistema
anteriormente."
      return True
  ipAux = ip.split('.')
  if not len(ipAux) == 4:
       print "La IP introducida no es correcta."
       print
       return False
  else:
       if (ipAux[0] < '1' or ipAux[0] > '255') or (ipAux[1] < '1' or ipAux[1] > '255') or
(ipAux[2] < '1' or ipAux[2] > '255') or (ipAux[3] < '1' or ipAux[3] > '255'):
            print "La IP introducida no es correcta."
            print
            return False
       else:
            print "La IP introducida no se encuentra en el sistema. ¿Quiere bloquearla
igualmente? (s/n):"
            sel = raw_input()
            if sel == 's':
                 return True
            elif sel == 'n':
                 return False
            else:
                 print "Opción no válida. No se procederá a bloquear la IP introducida."
                 return False

def selectionMenu(selection, attackersDict, reglas):
   if selection == '1':
       updateAttacks(attackersDict, reglas)
       updateMenu(reglas)
   elif selection == '2':
       consultAttacks(attackersDict, reglas)
   elif selection == '3':
      attacksLib.cleanRules(attackersDict, reglas)
       return
   elif selection == '4':
       attacksLib.deleteAllIptables(reglas)
   elif selection == '5':
       attacksLib.deleteM1(reglas)
   elif selection == '6':
       attacksLib.deleteM2(reglas)
   elif selection == '7':
       return
   else:
       print "Opcion no válida"
       print

def updateMenu(reglas):
```

```
    while True:
        print
        print "Para salir de este modo pulse c"
        print
        sel=raw_input()
        if sel == 'c':
                stopThread(reglas)
                break

def consultAttacks(attackersDict, reglas):
    while True:
        printPrimaryMenu()
        selection = raw_input()
        if selection == '1':
                while True:
                        printFirstMenu()
                        selection2=raw_input()
                        if selection2 == '1':
                        aut=[ip for ip in reglas if reglas[ip].typeRule == 'TMP' or
reglas[ip].active == 'False']
                        print aut[:20]
                        print
                        elif selection2 == '2':
                                aut=[ip for ip in reglas if not reglas[ip].typeRule == 'TMP' and
reglas[ip].active == 'True']
                           print aut[:20]
                           print
                        elif selection2 == '3':
                        printMenuBlockIp()
                                ip = raw_input()
                                check = checkIp(ip, reglas)
                                if check:
                                        attacksLib.blockIp(attackersDict, reglas, ip)
                        elif selection2 == '4':
                                printMenuDeleteRule()
                                ip = raw_input()
                                if ip in reglas.keys():
                                        attacksLib.deleteRule(ip, reglas)
                                        print "Regla eliminada."
                                else:
                                        print "La IP introducida es erronea o no se encuentra en el
sistema."
                        elif selection2 == '5':
                        break
                        else:
                                print
                                print "Opción no válida"
                                print
        elif selection == '2':
        while True:
```

```
                printSecondMenu()
                selection2=raw_input()
                if selection2 == '1':
                        aut=[reglas[key].rule for key in reglas if
attackersDict[key].blockBy=='U' and reglas[key].active == 'True']
                print aut[:20]
                print
                elif selection2 == '2':
                aut=[reglas[key].rule for key in reglas if attackersDict[key].blockBy=='I'
and reglas[key].active == 'True']
                print aut[:20]
                print
                elif selection2 == '3':
                break
                else:
                print
                print "Opción no válida"
                print
        elif selection == '3':
                distRules =  attacksLib.getDistCountry([ip for ip in reglas if reglas[ip].active
== 'True'])
                exportDistCountry(distRules)
                printDistCountry(distRules)
        elif selection == '4':
                printDistBlock(reglas)
        elif selection == '5':
        break
        else:
        print "Opción no válida"
        print

''' Funcion que imprime la distribucion de la reglas por paises
'''
def printDistCountry(dist):
        print
        print "-------------------------------------------------------------"
        print "||           DISTRIBUCION DE REGLAS POR PAIS            ||"
        print "-------------------------------------------------------------"
        print "||          COUNTRY          ||       Nº ATTACKS      ||"
        print "-------------------------------------------------------------"
        space="                          "
        lenSpace = len(space)
        space2 = "                       "
        lenSpace2 = len(space2)
        for distRule in dist:
                lenP1 = len(distRule[0])
                lenP2 = len(str(distRule[1]))
                numSpaceFin = (lenSpace - lenP1)
                numSpaceDer = (numSpaceFin//2)
                if numSpaceFin % 2:
```

```
                numSpaceIzq = numSpaceDer + 1
            else:
                numSpaceIzq = numSpaceDer
            lineFin = "||" + space[:numSpaceDer] + distRule[0] + space[:numSpaceIzq]

            numSpaceFin = (lenSpace2 - lenP2)
          numSpaceDer = (numSpaceFin//2)
          if numSpaceFin % 2:
              numSpaceIzq = numSpaceDer + 1
          else:
              numSpaceIzq = numSpaceDer
            lineFin = lineFin + "||" + space[:numSpaceDer] + str(distRule[1]) +
space[:numSpaceIzq] + "||"
            print lineFin
            print "----------------------------------------------------------"

''' Funcion que imprime la distribucion por tipo de bloqueo
'''
def printDistBlock(reglas):
     print
     print "----------------------------------------------------------"
     print "||       DISTRIBUCION DE REGLAS POR TIPO BLOQUEO        ||"
     print "----------------------------------------------------------"
     print "||        TYPE BLOCK          ||       Nº ATTACKS     ||"
     print "----------------------------------------------------------"
      space = "                    "
      lenSpace = len(space)
      line =  "||            M1              "
      lenM1 = len([ip for ip in reglas if reglas[ip].typeRule == 'M1' and reglas[ip].active
== 'True'])
      numSpaceFin = (lenSpace - len(str(lenM1)))
     numSpaceDer = (numSpaceFin//2)
     if numSpaceFin % 2:
          numSpaceIzq = numSpaceDer + 1
     else:
         numSpaceIzq = numSpaceDer
     line = line + "||" + space[:numSpaceDer] + str(lenM1) + space[:numSpaceIzq] + "||"
      print line
      print "----------------------------------------------------------"
      line =  "||            M2              "
     lenM1 = len([ip for ip in reglas if reglas[ip].typeRule == 'M2' and reglas[ip].active ==
'True'])
     numSpaceFin = (lenSpace - len(str(lenM1)))
     numSpaceDer = (numSpaceFin//2)
     if numSpaceFin % 2:
         numSpaceIzq = numSpaceDer + 1
     else:
         numSpaceIzq = numSpaceDer
     line = line + "||" + space[:numSpaceDer] + str(lenM1) + space[:numSpaceIzq] + "||"
     print line
```

```
        print "-------------------------------------------------------------"
        line =  "||            TMP              "
        lenM1 = len([ip for ip in reglas if reglas[ip].typeRule == 'TMP' and reglas[ip].active
== 'True'])
        numSpaceFin = (lenSpace - len(str(lenM1)))
        numSpaceDer = (numSpaceFin//2)
        if numSpaceFin % 2:
                numSpaceIzq = numSpaceDer + 1
        else:
                numSpaceIzq = numSpaceDer
        line = line + "||" + space[:numSpaceDer] + str(lenM1) + space[:numSpaceIzq] + "||"
        print line
        print "-------------------------------------------------------------"
        line =  "||           BY USER           "
        lenM1 = len([ip for ip in reglas if reglas[ip].typeRule == ' ' and reglas[ip].active ==
'True'])
        numSpaceFin = (lenSpace - len(str(lenM1)))
        numSpaceDer = (numSpaceFin//2)
        if numSpaceFin % 2:
                numSpaceIzq = numSpaceDer + 1
        else:
                numSpaceIzq = numSpaceDer
        line = line + "||" + space[:numSpaceDer] + str(lenM1) + space[:numSpaceIzq] + "||"
        print line
        print "-------------------------------------------------------------"


''' Funcion que exporta la distribución por pais
'''
def exportDistCountry(dist):
        f = open(os.path.dirname(os.path.abspath(__file__))+'/distCountry.txt', 'w')
        for country in dist:
                print>>f, country[0]+";"+str(country[1])
          f.close()
```

E Diagrama de clases

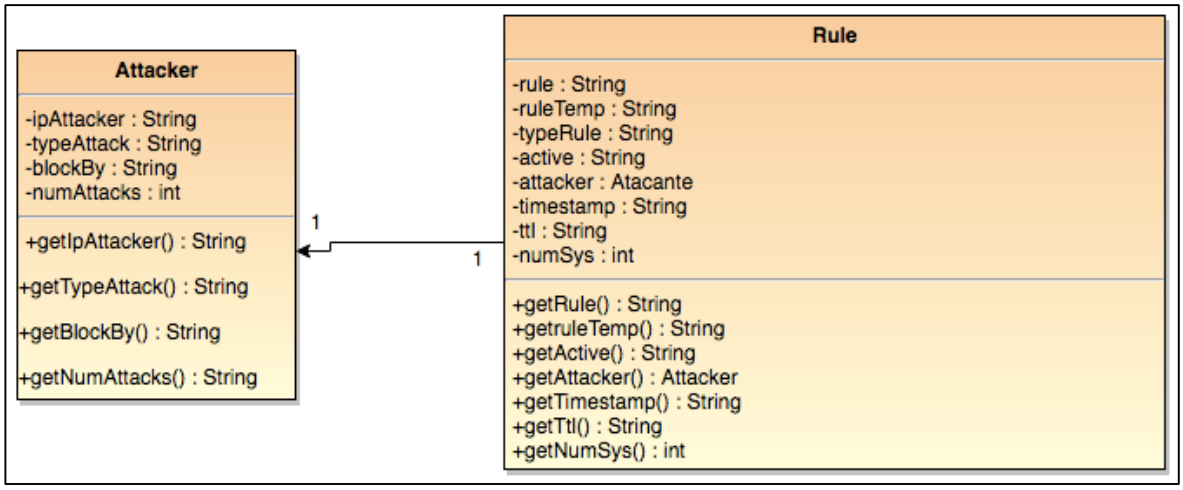


Ilustración 50. Diagrama de clases infraestructura.

F Diagrama funcional

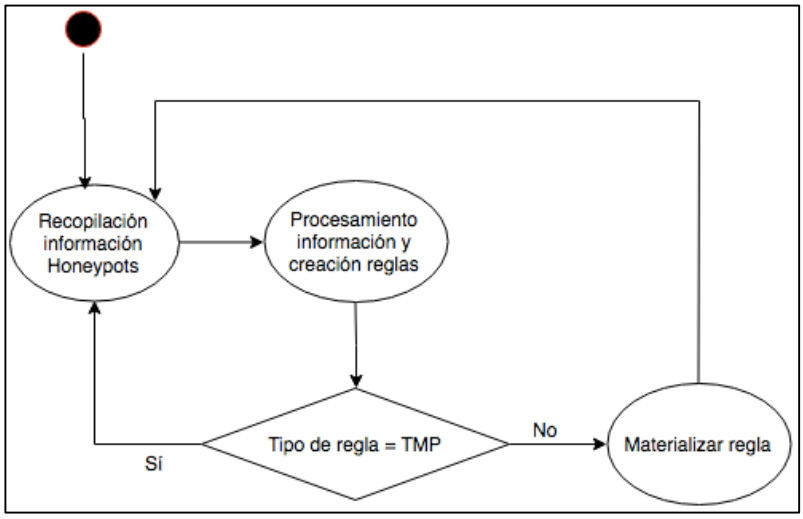


Ilustración 51. Diagrama funcional.

G Manual de uso del programa creado

En primer lugar deberemos tener creada un carpeta llamada logs donde tengamos los diferentes códigos que se necesitan para la correcta ejecución del programa.

A continuación introducimos en la línea de comandos la siguiente sentencia:

python HoneyPRules.py

Seguidamente nos a parecerá un pantalla en la cual podremos elegir entre las siguientes opciones:

1. Ejecutar actualización reglas.
2. Consultar información de las reglas y ataques.
3. Limpiar reglas de Ips inactivas.
4. Eliminar todas las reglas del sistema.

5. Salir.

La primera opción entrará en modo online, es decir modo actualización de reglas continuamente cada treinta minutos, pudiéndose configurar, dejando registrados los cambios sufridos en las reglas en los ficheros de log que genera la infraestructura. La tercera y la cuarte opción están relacionadas con las reglas que se han incorporado al ejecutar el anterior comando, una para limpiar aquellas reglas de las IP de los atacantes que llevan un periodo de tiempo sin realizar ataques y la otra para eliminar todas las reglas que se encuentran activas en el sistema, respectivamente.

Si introducimos la opción dos pasaremos a un nuevo menú, mostrando diferentes acciones que podremos realizar que exponemos a continuación:

1. Incorporar o eliminar una regla.
2. Ver reglas creadas.
3. Ver distribución reglas por país.
4. Ver distribución reglas por tipo de bloqueo.
5. Volver atrás.

Todas las opciones de este menú están relacionadas con visualizar información sobre las reglas que hay en el sistema menos la primera y la última. En la tercera y cuarta mostrará por pantalla una tabla con lo indicado en su título, es decir, mostrará el número de reglas existentes en el sistema y el país relacionado o el tipo de bloqueo que se ha generado. En la segunda opción podremos ver la reglas que han sido creadas tanto por parte de la infraestructura de forma automática como aquellas que el usuario haya introducido manualmente, mostrándose por pantalla un máximo de veinte debido a la posible gran cantidad de reglas que se puedan haber creado.

Si seleccionamos la primera opción se nos presentará un nuevo menú en el cual podremos introducir o eliminar reglas de nuestro sistema. El menú presenta la siguiente estructura:

1. Ver Ips sin bloquear en el sistema (máximo 20).
2. Ver Ips bloqueadas por el sistema y manuales. (máximos 20)
3. Introducir IP a bloquear.
4. Eliminar una regla.
5. Volver atrás.

En las dos primeras opciones se mostrarán por pantalla aquellas direcciones IP de atacantes que se encuentran en nuestra infraestructura, en la primera aquellas que todavía no han sido bloqueadas (tipo de regla asociado TMP) y en la segunda aquellas que sí lo han hecho, ya sea porque el tipo de regla asociado sea M1 o hayan sido bloqueadas anteriormente por el usuario. Al igual que cuando se muestran las reglas presentes en sistema, se mostrarán un máximo de veinte.

Las dos siguientes acciones están relacionadas con la incorporación o eliminación de reglas en nuestro sistema. Al introducir la opción tres o cuatro, nos pedirá que introduzcamos la dirección IP a ser bloqueada o eliminada en el formato XXX.XXX.XXX.XXX, donde XXX es un número entre 0 y 255, y una vez introducida quedará agregada o eliminada del sistema. Estas opciones no solo se limitan a direcciones IP que se encuentren en nuestro sistema, por lo que un usuario podrá bloquear una dirección IP que desee.

Una vez el usuario ya ha finalizado todas las acciones que quería realizar podrá navegar hasta el primer menú mencionado en este manual para salir. Al realizar este proceso el sistema dará la posibilidad de guardar todos los cambios realizados por el usuario para que

sean cargados en la siguiente ejecución del programa debido a que se eliminan todas generadas por la infraestructura al finalizar la ejecución del programa.